MinFinder: Locating all the local minima of a function

Ioannis G. Tsoulos, Isaac E. Lagaris*

Department of Computer Science, University of Ioannina,
P.O. Box 1186, Ioannina 45110 - GREECE

Abstract

A new stochastic clustering algorithm is introduced that aims to locate all the local minima of a multidimensional function inside a bounded domain. The accompanying software (MinFinder) is written in ANSI C++. However, the user may code his objective function either in C++, C or Fortran77. We compare the performance of this new method to the performance of Multistart and Topographical Multileve Single Linkage Clustering on a set of benchmark problems.

PACS:02.60.-x ; 02.60.Pn ; 07.05.Kf; 02.70.Lq

Keywords: Global optimization, stochastic methods, Monte Carlo, clustering, region of attraction.

PROGRAM SUMMARY

Title of program: MinFinder

Catalogue identifier:

Program available from: CPC Program Library, Queen's University of Belfast, N. Ireland.

Computer for which the program is designed and others on which it has been tested: The tool is designed to be portable in all systems running the GNU C++ compiler.

Installation: University of Ioannina, Greece.

Programming language used: GNU-C++, GNU-C, GNU Fortran - 77.

Memory required to execute with typical data: 200KB.

*Corresponding author e-mail: lagaris@cs.uoi.gr

No. of bits in a word: 32

No. of processors used: 1

Has the code been vectorised or parallelized?: No.

No. of bytes in distributed program,including test data etc.: 300 Kbytes.

Distribution format: gzipped tar file.

Keywords: Global optimization, stochastic methods, Monte Carlo, clustering, region of attraction.

Nature of physical problem: A multitude of problems in science and engineering are often reduced to minimizing a function of many variables. There are instances that a local optimum does not correspond to the desired physical solution and hence the search for a better solution is required. Local optimization techniques can be trapped in any local minimum. Global optimization is then the appropriate tool. For example, solving a non - linear system of equations via optimization, one may encounter many local minima that do not correspond to solutions, i.e. they are far from zero.

Typical running time: Depending on the objective function.

LONG WRITE UP

1 Introduction

Locating the local minima of a multidimensional function $f(x)$, appears frequently as a subproblem in a variety of scientific applications. Among the several methods employed to treat this problem, stochastic methods seem to be the most popular, most probably due to both their effectiveness and implementation simplicity. An important subclass of stochastic methods are the so called clustering techniques, pioneered by Becker and Lago [1], Törn [2], Boender et. al. [3], Rinnooy Kan and Timmer [4, 5]. Clustering techniques are based on the "multistart" algorithm and their goal is to limit the number of local search applications. A cluster is defined as a set of points that are believed to belong to the basin of attraction of the same minimum, and hence only one local search is (optimally) required to locate it. The basin of attraction of a local minimum x^* is defined as:

$$A\left(x^*\right) = \{x : x \in S \subset R^n,\ L(x) = x^*\}$$

where $L(x)$ is the point where the local search procedure L terminates when started at point x. Here L is supposed to be a deterministic local optimization method such as BFGS [8], Steepest Descent, Modified Newton etc. The present work is a clustering technique based on the "Topographical Multi Level Single

Linkage" (TMLSL) of Ali and Storey [6]. The modifications we present are important and render the technique significantly more efficient.

In section 2 we present the proposed algorithm and in section 3 we present the results of numerical experiments, along with our conclusions. In section 4 we layout the documentation of the related software, describing its distribution, installation and use. In the appendix we list the test functions employed in our experiments for evaluating the performance of the proposed scheme.

2 Description of the algorithm

The proposed algorithm is based heavily on three key elements: the typical distance, the gradient criterion and the Double Box stopping rule.

2.1 Typical distance

A clustering procedure forms clusters of points by measuring the distance of a candidate point from the estimated center of the cluster. This distance is checked against a threshold and a decision is made accordingly. The algorithm uses a typical distance defined by:

$$r_t \equiv \frac{1}{M} \sum_{i=1}^{M} \left| x_i - L\left(x_i\right) \right| \tag{1}$$

where x_i are starting - points for the local search procedure L, and M is the number of the performed local searches. The main idea behind equation (1) is that after a number of iterations and a number of local searches the quantity r_t will be a reasonable approximation for the mean radius of the regions of attraction. To see this note that if we denote by M_l the number of times that the local search procedure discovered the minimizer x_l^*, then a basin radius may be defined as:

$$R_l = \frac{1}{M_l} \sum_{j=1}^{M_l} \left| x_l^{(j)} - x_l^* \right| \tag{2}$$

where $\left\{ x_l^{(j)},\ j = 1, \ldots, M_l \right\} = \left\{ x_i,\ i = 1, \ldots, M \right\} \cap A\left(x_l^*\right)$, i.e. $L\left(x_l^{(j)}\right) = x_l^*$. Since by definition $M = \sum_{l=1}^{w} M_l$, where w is the number of local minima discovered so far, a mean basin radius may be defined as:

$$< R > \equiv \sum_{l=1}^{w} \frac{M_l}{M} R_l = \frac{1}{M} \sum_{l=1}^{w} \sum_{j=1}^{M_l} \left| x_l^{(j)} - x_l^* \right| \tag{3}$$

Comparing eqs. (1), (2) and (3), it follows that $r_t =< R >$.

2.2 Gradient criterion

The value of the objective function $f(x)$ at a point x in the neighborhood of a local minimum x^*, can be estimated as:

$$f(x) \simeq f\left(x^*\right) + \frac{1}{2}\left(x - x^*\right)^T B^*\left(x - x^*\right) \tag{4}$$

where B^* is the Hessian matrix at the minimum x^*. By applying the gradient operator in each part of the equation (4) we obtain:

$$\nabla f(x) \simeq B^*\left(x - x^*\right) \tag{5}$$

In the same way for any other point y, in the neighborhood of a local minimum x^*, we have:

$$\nabla f(y) \simeq B^*\left(y - x^*\right) \tag{6}$$

By subtracting (5) from (6) and by multiplying from the left with $(x-y)^T$ we obtain:

$$(x-y)^T\left(\nabla f(x) - \nabla f(y)\right) \simeq (x-y)^T B^*(x-y) > 0 \tag{7}$$

(since B^* is positive definite).

2.3 Double Box stopping rule

The most widely used stopping rule is that developed by Rinnooy Kan and Timmer [5], where the algorithm stops iterating if

$$\frac{w(M-1)}{M-w-2} < w + \frac{1}{2} \tag{8}$$

M being the number of total sample points and w the number of the located so far. The relation (8) may be rewritten as:

$$M > 2w^2 + 3w + 2 \tag{9}$$

which means that in order to stop iterating the number of sample points must be greater than the square of the located local minima. This is undesirable for functions with many local minima and a more effective stopping rule must be employed. In the proposed method we use a termination rule that estimates the uncovered portion of the search space. A relative measure for this may be given by:

$$U = 1 - \sum_{i=1}^{w} \frac{m\left(A_i\right)}{m(S)} \tag{10}$$

where w is the number of the discovered so far local minima. The quantity $\frac{m(A_i)}{m(S)}$ may be approximated by the fraction $\frac{T_i}{T}$, where T_i is the sum of local searches that have ended up at the local minimum x_i^*, plus the sample points that have been allocated to the cluster centered at x_i^*, and T is the total number of sample points. Hence an approximation for U may be given by:

$$U \simeq \tilde{U} = 1 - \sum_{i=1}^{w} \frac{T_i}{T} \tag{11}$$

Unfortunately, equation (11) always yields $\tilde{U} = 0$ and hence the uncovered space can not be estimated with the above relation. However, a larger box S_2, that contains S, is constructed in a way such that $m(S_2) = 2 \times m(S)$, $m(S)$ being the Lebesgue measure of S. A unique (fake) local minimum is considered to be contained in $A_0 = S_2 - S$ with measure $m(A_0) = m(S)$. The uncovered portion of the search space is now given by:

$$U = 1 - \frac{\sum_{i=0}^{w} m(A_i)}{m(S_2)} = 1 - \frac{m(A_0)}{m(S_2)} - \frac{\sum_{i=1}^{w} m(A_i)}{m(S_2)} = \frac{1}{2} - \frac{\sum_{i=1}^{w} m(A_i)}{m(S_2)} \tag{12}$$

The quantity $\frac{m(A_i)}{m(S_2)}$ is approximated again by the the fraction $\frac{T_i}{T}$, T being the total number of sample points in S_2 and hence:

$$U \simeq \tilde{U} = \frac{1}{2} - \sum_{i=1}^{w} \frac{T_i}{T} \tag{13}$$

At every iteration we sample points from S_2 until we have collected N points belonging to S. After k iterations the total number of sample points are M_k and the quantity

$$\delta_k \equiv \frac{kN}{M_k} \tag{14}$$

has an expectation value

$$< \delta >_k = \frac{1}{k} \sum_{i=1}^{k} \delta_i \tag{15}$$

that asymptotically tends to $\frac{m(S)}{m(S_2)} = \frac{1}{2}$. The variance is given by

$$\sigma_k^2(\delta) = < \delta^2 >_k - < \delta >_k^2 \tag{16}$$

and tends to zero as $k \to \infty$. This is a smoother quantity than the expectation value and better suited for the following termination procedure:

1. Initially set $a = 0$ and $k = 0$.
2. Sample from S_2, until N points fall in S.
3. Calculate the quantity $\delta_k = \frac{kN}{M_k}$.
4. Calculate the expectation value of δ_k.
5. Calculate the deviation $\sigma(\delta)$.

6. Perform a step of the MinFinder algorithm (described in subsection 2.4).
7. If one or more new minima are found, set $a = p\sigma^2(\delta)$ and repeat from step 2.
8. If $\sigma^2(\delta) < a$, TERMINATE, otherwise repeat from step 2.

The parameter p is in the range $(0, 1)$. For small values of p ($p \to 0$) the algorithm searches the area exhaustively, while for $p \to 1$ the algorithm terminates earlier, but perhaps prematurely. As a compromise between exhaustive and speed search, we suggest the value of $p = 0.5$.

2.4 Steps of the algorithm

In the following by X^* we denote the set of the local minima collected so far. Initially $X^* = \emptyset$. The steps of the proposed algorithm are as:

Checking Step:

- **Set** $V = \emptyset$.
- Set T as the set of N points sampled from the Double Box procedure.
- **Forall** $x \in T$ **do**
  - Check if x is a valid starting point (validated by a procedure described later) , and if so add it to V.
- **Endforall**

Enrichment Step:

- **If** $\frac{|V|}{N} < \frac{1}{2}$ **Then**
  - $N = \min\left(N + \frac{N}{10}, \text{NMAX}\right)$, where NMAX is a predefined upper limit for the number of samples in each generation. This step prevents the algorithm form an insufficient exploration of the search space. The typical value for this parameter is NMAX=100.
- **Endif**

Main Step:

- **Forall** $x \in V$ **do**
  - **If** x is considered as start point (the validation procedure is once more performed, because a point that was considered as a start point earlier may no longer be due to the presence of new local minima) **Then**

        * **Start** a local search $y = L(x)$.
        * **Compute** the typical distance r_t using equation (1).
        * **If** $y \notin X^*$ **Then**
            · **Set** $X^* = X^* \cup y$.
        * **Endif**
    - **Endif**

- **Endforall**

A point x is to be considered as start point if none of the following conditions holds:

- There is an already located minimum z that satisfies the conditions
    1. $(x - z)^T(\nabla f(x) - \nabla f(z)) > 0$.
    2. $|x - z| < \min_{i,j\ i \neq j} |z_i - z_j|\,,\ z_i \in X^*,\ z_j \in X^*$.
- x is near to another point $y \in V$ that satisfies the conditions
    1. $|x - y| < r_t$.
    2. $(x - y)^T(\nabla f(x) - \nabla f(y)) > 0$.

3 Experimental results

We compared the new method against Multistart and Topographical Multilevel Single Linkage [6]. We have used the Double Box stopping rule in all methods. All experiments have been repeated 50 times with different random number generator seed. The initial sample size was set to 20. In the following tables the columns "PROBLEM", "MINIMA", "FOUND", "FEVALS", "GEVALS" and "TIME" denote the name of the objective function, the known number of local minima, the average number of the discovered local minima, the average number of function evaluations performed, the average number of the gradient evaluations and the average CPU time. All the experiments were run on an AMD ATHLON 2400+ with 256MB RAM running Slackware Linux v9.1. The local search procedure used in all methods was a BFGS implementation due to Powell [7].

In tables 1, 2 and 3 we list the experimental results from the methods Multistart, TMLSL and Minfinder correspondingly. The results indicate that the proposed method is capable of finding almost all the minima of an objective function with less effort than other stochastic methods such as Multistart and TMLSL. The provided software has a very simple installation procedure and it can be easily installed in any UNIX operating system equipped with any ANSI C++ compiler.

Table 1: Multistart results

PROBLEM	MINIMA	FOUND	FEVALS	GEVALS	TIME
CAMEL	6	6	11138	10741	0.04
RASTRIGIN	49	49	17714	16989	0.06
SHUBERT	400	400	557668	535368	6.44
GRIEWANK2	529	529	1697081	1646952	20.78
HANSEN	527	527	586090	563131	8.71
GKLS(3,60)	60	59	781378	691787	22.98
GUILIN(5,50)	50	50	195392	190566	1.76
GUILIN(10,50)	50	50	324645	318989	6.71
BRANIN	3	3	7488	7107	0.04
GOLDSTEIN	4	4	15951	15570	0.08
SHEKEL5	5	5	25930	25431	0.75
SHEKEL7	7	7	28946	28382	1.11
SHEKEL10	10	10	30808	30175	1.09
HARTMAN3	3	3	11467	11086	0.14
HARTMAN6	2	2	16145	15764	0.52

Table 2: TMLSL results

PROBLEM	MINIMA	FOUND	FEVALS	GEVALS	TIME
CAMEL	6	6	3486	2643	0.03
RASTRIGIN	49	49	23809	12195	0.78
SHUBERT	400	400	337476	56597	127.47
GRIEWANK2	529	528	2233048	1840673	251.57
HANSEN	527	527	838554	130642	451.38
GKLS(3,60)	60	56	880641	357247	53.31
GUILIN(5,50)	50	50	220678	201519	3.21
GUILIN(10,50)	50	50	426882	401254	10.57
BRANIN	3	3	1036	631	0.01
GOLDSTEIN	4	4	4647	3871	0.04
SHEKEL5	5	5	7554	6366	0.23
SHEKEL7	7	6	11546	9936	0.43
SHEKEL10	10	9	18649	15786	0.65
HARTMAN3	3	3	1837	1559	0.03
HARTMAN6	2	2	860	621	0.03

Table 3: Minfinder results

PROBLEM	MINIMA	FOUND	FEVALS	GEVALS	TIME
CAMEL	6	6	1598	2187	0.02
RASTRIGIN	49	49	1723	2975	0.08
SHUBERT	400	400	17404	41849	7.07
GRIEWANK2	529	529	1035094	1190595	80.16
HANSEN	527	527	60916	94382	15.79
GKLS(3,60)	60	56	169280	297160	13.89
GUILIN(5,50)	50	50	84675	88111	1.21
GUILIN(10,50)	50	50	173186	179397	4.40
BRANIN	3	3	498	604	0.01
GOLDSTEIN	4	4	2197	2364	0.02
SHEKEL5	5	5	7144	7365	0.25
SHEKEL7	7	7	17125	17377	0.75
SHEKEl10	10	10	21551	21661	0.90
HARTMAN3	3	3	1581	1737	0.03
HARTMAN6	2	2	1090	1194	0.05

4 Software documentation

4.1 Distribution

The package is distributed in a tar.gz file named `MinFinder.tar.gz` and under UNIX systems the user must issue the following commands to extract the associated files:

1. gunzip `MinFinder.tar.gz`
2. tar xfv `MinFinder.tar`

These steps create a directory named `MinFinder` with the following contents:

1. **bin**: A directory which is initially empty. After compilation of the package, it will contain the executable **make_program**.
2. **doc**: This directory contains the documentation of the package (this file) in different formats: A LyX file, A LaTeX file and a PostScript file.
3. **examples**: A directory that contains the test functions used in this article, written in ANSI C++ and the Fortran77 version of the Six Hump Camel function.
4. **include**: A directory which contains the header files for all the classes of the package.

5. **src**: A directory containing the source files of the package.

6. **Makefile**: The input file to the `make` utility in order to build the tool. Usually, the user does not need to change this file.

7. **Makefile.inc**: The file that contains some configuration parameters, such as the name of the C++ compiler etc. The user must edit and change this file before installation.

4.2 Installation

The following steps are required in order to build the tool:

1. Uncompress the tool as described in the previous section.

2. `cd MinFinder`

3. Edit the file `Makefile.inc` and change (if needed) the five configuration parameters.

4. Type `make`.

The five parameters in `Makefile.inc` are the following:

1. **CXX**: It is the most important parameter. It specifies the name of the C++ compiler. In most systems running the GNU C++ compiler this parameter must be set to g++.

2. **CC**: If the user written programs are in C, set this parameter to the name of the C compiler. Usually, for the GNU compiler suite, this parameter is set to gcc.

3. **F77**: If the user written programs are in Fortran 77, set this parameter to the name of the Fortran 77 compiler. For the GNU compiler suite a usual value for this parameter is g77.

4. **F77FLAGS**: The compiler GNU FORTRAN 77 (g77) appends an underscore to the name of all subroutines and functions after the compilation of a Fortran source file. In order to prevent this from happening we can pass some flags to the compiler. Normally, this parameter must be set to -fno-underscoring.

5. **ROOTDIR**: Is the location of the MinFinder directory. It is critical for the system that this parameter is set correctly. In most systems, it is the only parameter which must be changed.

4.3 User written subprograms

In figure 1 we see the template of the objective function in the C programming language. The same scheme is used also in C++, but the code has the line

```
extern ''C'' {
```

before the functions and the line

```
}
```

after them, in order to prevent the compiler from generating symbols that will not cause problem to the linking process. The template for Fortran 77 is given in figure 2. The symbol d denotes the dimension of the objective function. The meaning of the functions are the following:

1. **getdimension**(): It returns the dimension of the objective function.

2. **getleftmargin**(left): It fills the double precision array left with the left margins of the objective function.

3. **getrightmargin**(right): It fills the double precision array right with the right margins of the objective function.

4. **funmin**(x): It returns the value of the objective function evaluated at point x.

5. **granal**(x,g): It returns in a double precision array g the gradient of the objective function at point x.

4.4 The utility make_program

After the compilation of the package, the executable `make_program` will be placed in the subdirectory `bin` in the distribution directory. This program creates the final executable and it takes as its only argument the name of the file containing the objective function. The utility checks the suffix of the file and it uses the appropriate compiler. If this suffix is .cc or .c++ or .CC or .cpp, then it invokes the C++ compiler. If the suffix is .f or .F or .for then it invokes the Fortran 77 compiler. Finally, if the suffix is .c it invokes the C compiler.

4.5 The utility MinFinder

After the compilation of the objective function with the tool `make_program` the executable `MinFinder` is created. This executable can take the following arguments in the command line:

1. **-h**: The program prints a help screen to the user and stops.

Figure 1: Formulation in C

```
int getdimension()
{
}

void getleftmargin(double *left)
{
}

void getrightmargin(double *right)
{
}

double funmin(double *x)
{
}

void granal(double *x,double *g)
{
}
```

2. **-s** `size`: The integer parameter `size` is used as the size of the sample. (i.e. $N = \text{size}$) The default value for this parameter is 20.

3. **-o** `filename`: The string parameter `filename` specifies a file where all the discovered local minima will be disposed after the termination of the program.

4. **-p** `level`: The integer parameter `level` can take only two values: 0 or 1. If the value is 0, then no output will be sent to the standard output. If the value is 1, then after each iteration, the algorithm prints a line displaying the number of iterations, the number of located minima, the total number of function calls, the total number of gradient calls, the value of $\sigma^2(\delta)$ and the value of a used in the Double Box stopping rule. The default value for this parameter is 0.

5. **-r** `seed`: The integer parameter `seed` specifies the seed for the random number generator. It can assume any integer value.

4.6 A working example

Consider the Six Hump Camel function given by

$$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,\ x \in [-5,5]^2$$

Figure 2: Formulation in Fortran 77

integer function getdimension()
getdimension = d
end

subroutine getleftmargin(left)
double precision left(d)
end

subroutine getrightmargin(right)
double precision right(d)
end

double precision function funmin(x)
double precision x(d)
end

subroutine granal(x,g)
double precision x(d)
double precision g(d)
end

with 6 local minima. The implementation of this function in C++ and in Fortran77 is shown in figures 3 and 4. Let the file with the C++ code be named `camel.cc` and that with the Fortran code `camel.f.` Let these files be located in the `examples` subdirectory. Change to the `examples` subdirectory and create the `MinFinder` executable with the `make_program` command:

```
../bin/make_program camel.cc
```

or for the Fortran 77 version

```
../bin/make_program camel.f
```

The `make_program` responds:

```
RUN ./MinFinder IN ORDER TO RUN THE PROBLEM
```

Run `MinFinder` by issuing the command:

```
./MinFinder -o camel.out -p 1 -r 7
```

The resulting output appears as:

```
iters= 1  minimum= 1 fevals=   24 gevals=   33    delta=      0   stopat=1.9763e-323
iters= 2  minimum= 2 fevals=   89 gevals=  106    delta=      0   stopat=      0
```

Figure 3: Implementation of Camel function in C++.

```
extern "C"{
 int getdimension()
 {
   return 2;
 }

 void getleftmargin(double *left)
 {
   left[0]=-5.0;
   left[1]=-5.0;
 }

 void getrightmargin(double *right)
 {
   right[0]=5.0;
   right[1]=5.0;
 }

 double funmin(double *x)
 {
   double x1=x[0],x2=x[1];
   return 4*x1*x1-2.1*x1*x1*x1*x1+
      x1*x1*x1*x1*x1*x1/3.0+x1*x2-4*x2*x2+4*x2*x2*x2*x2;
 }

 void granal(double *x,double *g)
 {
   double x1=x[0],x2=x[1];
   g[0]=8*x1-8.4*x1*x1*x1+2*x1*x1*x1*x1*x1+x2;
   g[1]=x1-8*x2+16*x2*x2*x2;
 }
}
```

```
iters= 3  minimum= 3 fevals=  151 gevals=  178    delta=6.7063e-05 stopat=3.3531e-05
iters= 4  minimum= 4 fevals=  236 gevals=  272    delta=5.4213e-05 stopat=2.7106e-05
iters= 5  minimum= 6 fevals=  282 gevals=  331    delta=4.1245e-05 stopat=2.0622e-05
iters= 6  minimum= 6 fevals=  386 gevals=  446    delta=3.5875e-05 stopat=1.7938e-05
iters= 7  minimum= 6 fevals=  520 gevals=  591    delta=3.0131e-05 stopat=1.7938e-05
iters= 8  minimum= 6 fevals=  604 gevals=  688    delta=2.5983e-05 stopat=1.7938e-05
iters= 9  minimum= 6 fevals=  680 gevals=  778    delta=2.2972e-05 stopat=1.7938e-05
iters= 10 minimum= 6 fevals=  804 gevals=  916    delta=2.0464e-05 stopat=1.7938e-05
iters= 11 minimum= 6 fevals=  888 gevals= 1015    delta=1.8794e-05 stopat=1.7938e-05
```

All minima are discovered by iteration 5, however the program continued until iteration 11, because `delta`, which corresponds to the quantity $\sigma^2(\delta)$, at the 5^{th} iteration was not lower than `stopat` (the quantity a in our algorithm). The discovered minima are written to the file `camel.out`, the contents of which are listed below:

```
2
6
-1.703606715    0.7960835687   -0.2154638244
```

Figure 4: Implementation of Camel function in Fortran 77.

```
integer function getdimension()
getdimension = 2
end

subroutine getleftmargin(left)
double precision left(2)
left(1)=-5.0
left(2)=-5.0
end

subroutine getrightmargin(right)
double precision right(2)
right(1)= 5.0
right(2)= 5.0
end

double precision function funmin(x)
double precision x(2)
double precision x1,x2
x1=x(1)
x2=x(2)
funmin=4*x1**2-2.1*x1**4+x1**6/3.0+x1*x2-4*x2**2+4*x2**4
end

subroutine granal(x,g)
double precision x(2)
double precision g(2)
double precision x1,x2
x1=x(1)
x2=x(2)
g(1)=8.0*x1-8.4*x1**3+2*x1***5+x2;
g(2)=x1-8.0*x2+16.0*x2**3;
end
```

```
 0.0898420131  -0.712656403    -1.031628453
-0.0898420131   0.712656403    -1.031628453
-1.607104753   -0.5686514549    2.10425031
 1.703606715   -0.7960835687   -0.2154638244
 1.607104753    0.5686514549    2.10425031
```

In the first line the single entry (number 2) denotes the dimensionality of the problem. In the second line the single entry (number 6) denotes the number of the discovered local minima. In each of the following lines there are three entries. The first two correspond to the parameter values of the minimizer, while the third to the corresponding value of the objective function.

A. Test functions

We list the test functions used in our experiments, the associated search domains and the number of the known local minima.

Camel

$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$, $x \in [-5, 5]^2$ with 6 local minima.

Rastrigin

$f(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$, $x \in [-1, 1]^2$ with 49 local minima.

Shubert

$f(x) = -\sum_{i=1}^{2}\sum_{j=1}^{5} j\left\{\sin((j+1)x_i) + 1\right\}$,$x \in [-10, 10]^2$ with 400 local minima.

Griewank2

$f(x) = 1 + \frac{1}{200}\sum_{i=1}^{2} x_i^2 - \prod_{i=1}^{2}\frac{\cos(x_i)}{\sqrt{(i)}}$, $x \in [-100, 100]^2$ with 529 loca minima.

Hansen

$f(x) = \sum_{i=1}^{5} i\cos\left[(i-1)x_1 + i\right]\sum_{j=1}^{5} j\cos\left[(j+1)x_2 + j\right]$, $x \in [-10, 10]^2$ with 527 local minima.

Gkls

$f(x) = \text{Gkls}(x, n, w)$, is a function with w local minima, described in [10], $x \in [-1, 1]^n$, $n \in [2, 100]$. In our experiments we use $n = 3$ and $w = 60$.

Guilin Hills

$f(x) = 3 + \sum_{i=1}^{n} c_i \frac{x_i+9}{x_i+10} \sin\left(\frac{\pi}{1-x_i+\frac{1}{2k_i}}\right)$, $x \in [0,1]^n$, $c_i > 0$ and k_i are positive integers. This function has $\prod_{i=1}^{n} k_i$ local minima. In our experiments we use $n = 5, 10$ and we have arranged the values of k_i so that the number of minima was 50. These cases are entitled as GUILIN(5,50) and GUILIN(10,50) in the following tables.

Branin

$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$ with $-5 \le x_1 \le 10,\ 0 \le x_2 \le 15$. The function has 3 minima in the specified range.

GoldStein & Price

$$\begin{aligned} f(x) &= [1 + (x_1 + x_2 + 1)^2 \\ &\quad (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times \\ &\quad [30 + (2x_1 - 3x_2)^2 \\ &\quad (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned}$$

The function has 4 local minima in the range $[-2,2]^2$.

Shekel 5

$$f(x) = -\sum_{i=1}^{5} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

with $x \in [0,10]^4$ and

$$a = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{pmatrix}$$

and

$$c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \end{pmatrix}$$

The function has 5 local minima in the specified range.

Shekel 7

$$f(x) = -\sum_{i=1}^{7} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

with $x \in [0, 10]^4$ and

$$a = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 3 & 5 & 3 \end{pmatrix}$$

and

$$c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \end{pmatrix}$$

The function has 7 local minima in the specified range.

Shekel 10

$$f(x) = -\sum_{i=1}^{10} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

with $x \in [0, 10]^4$ and

$$a = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}$$

and

$$c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.6 \end{pmatrix}$$

Hartman 3

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right)$$

with $x \in [0,1]^3$ and

$$a = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$$

and

$$c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}$$

and

$$p = \begin{pmatrix} 0.3689 & 0.117 & 0.2673 \\ 0.4699 & 0.4387 & 0.747 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}$$

The function has 3 minima in the specified range.

Hartman 6

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right)$$

with $x \in [0,1]^6$ and

$$a = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$$

and

$$c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}$$

and

$$p = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$$

The function has 2 local minima in the specified range.

References

[1] Becker R. W. and Lago G.V, "A global optimization algorithm", in Proceedings of the 8^{th} Allerton Conference on Circuits and Systems Theory, 1970.

[2] Törn A. A., "A search clustering approach to global optimization", in Dixon L.C.W and Szegö (eds.), Towards Global Optimizations 2, North - Holland, Amsterdam, 1978.

[3] Boender C.G.E., Rinnooy Kan A.H.G , Timmer G.T and Stougie L., "A stochastic method for global optimization", Mathematical Programming **22**, pp. 125-140, 1982.

[4] Rinnooy Kan A.H.G. and Timmer G.T., "Stochastic global optimization methods, Part I: Clustering methods", Mathematical programming **39**, pp. 27-56, 1987.

[5] Rinnooy Kan A.H.G. and Timmer G.T., "Stochastic global optimization methods, Part II: Multi level methods", Mathematical Programming **39**, pp. 57-78, 1987.

[6] Ali M.M. and Storey C. "Topographical Multilevel Single Linkage", Journal of Global Optimization 5, 349-358, 1994.

[7] Powel M. J. D., "A Tolerant Algorithm for Linearly Constrained Optimization Calculations", Mathematical Programming 45 (1989), 547.

[8] Fletcher R., "A new approach to variable metric algorithms", Computer Journal **13**, 317-322, 1970.

[9] Boender C. G. E., Kan Rinnooy A. H. G, "Bayessian stopping rules for multistart global optimization methods", Mathematical Programming 37 (1987), 59-80.

[10] Gaviano M., Ksasov D. E., Lera D. and Sergeyev, Y. D. Software for generation of classes of test functions with known local and global minima for global optimization, ACM Trans. Math. Softw. **29**, pp. 469-480, 2003.