Implementation of novel methods of global and non-smooth optimization. GANSO programming library

G. Beliakov[1] and J. Ugon[2]

April 24, 2007

[1] *Faculty of Science and Technology, Deakin University, 221 Burwood Hwy, Burwood 3125, Australia.*
email: gleb@deakin.edu.au
[2] *School of Information Technology and Mathematical Sciences, University of Ballarat, Mt Helen, 3353 Victoria, Australia.*
email: j.ugon@ballarat.edu.au

Abstract

We discuss the implementation of a number of modern methods of global and non-smooth continuous optimization in a programming library `GANSO`. `GANSO` implements the Derivative-Free Bundle Method, the Extended Cutting Angle method, Dynamical System-based Optimization and their various combinations and heuristics. This article outlines the main ideas behind each method, illustrates the syntaxis and usage of the library, and reports on its interfacing with Matlab and Maple packages.

Keywords Global optimization, non-smooth optimization, nonlinear programming, numerical optimization.

1 Introduction

Large research effort in the field of optimization has resulted in a number of new powerful methods, capable of solving problems with very complicated objective functions. Notably, with the objective functions that are not differentiable, and those that possess a large number of local optima. Such functions frequently arise in applications, for example when using exact penalty functions in constrained optimization, or when calculating molecular structures (protein folding, docking studies, etc. [11, 17, 20, 21, 30, 32]). Derivative-free methods are becoming more popular, as exact gradients (and even more so, subgradients) are typically unknown in the applications.

Various heuristics have become very popular (e.g., simulated annealing and genetic programming), possibly because of their intuitive interpretation and easiness of implementation [43]. Even though sometimes their use is not warranted, as more sophisticated methods deliver substantially better results, there is generally lack of detailed comparative analysis (with a few exceptions, see [22, 31]), which lead practitioners to choosing less powerful but more user friendly methods and relying more on computational power.

One problem is that many modern methods are hidden beyond the wall of mathematical formulae, and are not easy to implement. In those case where implementation is available, it is not as generic and as thoroughly tested as commercial packages, nor it is straightforward to use by practitioners. A notable exception are the packages by M.J.D. Powell [37, 38] and LGO by J. Pinter [35].

The aim of our work was to collect a number of recent methods of global and non-smooth optimization, mainly those developed at the University of Ballarat by A. Rubinov's group, and implement them in a programming library with a generic user-friendly interface, which would be attractive to non-expert users. Our hope is that the easiness of use of such a library would persuade practitioners to use advanced optimization methods.

The result of this work is the programming library `GANSO` (which stands for Global And Non-Smooth Optimization), presented in this article [16]. `GANSO` implements four different approaches to global non-smooth optimization, and also their combinations. For unconstrained local non-smooth optimization it uses the Derivative Free Bundle Method (DFBM), based on finite difference approximation to the subgradient [2, 3]. For global non-convex optimization on some compact domain D it uses the Extended Cutting Angle method (ECAM) [4, 6–8, 41], a simple Random start method, and a method based on trajectories of Dynamical Systems (DSO) [24–26]. There are also several combinations of these methods.

One of the strengths of the GANSO library is the possibility to use several combinations of these methods. Research has shown that adequately combining two optimization methods can lead to greatly improved results [5, 10, 25]. In GANSO, we implement combinations that have already been shown to work well, and some novel approaches.

In this article we give an overview of the optimization methods implemented in `GANSO` , and discuss some of the combinations of global and local search. We illustrate the syntaxis and the usage of `GANSO` library on simple examples, and also report on some implementation issues related to interfacing `GANSO` with popular Matlab and Maple packages.

2 Optimization methods

`GANSO` library implements a number of methods of global and non-smooth optimization. These methods aim at solving the following generic optimization problem

$$\begin{gathered}\text{minimize } f(x)\\ \text{subject to } x \in D \subseteq R^n. \end{gathered} \tag{1}$$

The feasible domain D is specified by a number of linear constraints (equalities and inequalities), including box constraints

$$a_i \leq x_i \leq b_i \; i = 1, \dots, n.$$

In the case of unconstrained minimization, $D = R^n$.

The class of objective functions f dealt with in `GANSO` is very broad. We do not assume differentiability of f, and only require its Lipschitz continuity (local or global) for ECAM and DFBM. DSO method can deal with discontinuous functions. The Lipschitz continuity is expressed as

$$|f(x) - f(y)| \leq L d(x, y),$$

where x, y are two points in D, $d(x, y)$ is the distance between these points (e.g., Euclidean distance) and L is some positive number. The inequality should hold for all $x, y \in D$.

Under such a general condition, the optimization problem is extremely difficult. The objective function may have many local extrema, and locating its absolute minimum (global minimum) is very challenging indeed. Computation of the direction of descent at those points where f is not differentiable is also a challenge. Familiar methods, such as quasi-Newton, or conjugate gradient, may simply not work for this type of problems. There is substantial literature on the subject of global and non-smooth optimization. The references section lists a few popular books and overviews, e.g., [11, 17–19, 23, 29, 33, 36, 42].

`GANSO` implements three different approaches to global non-smooth optimization, and also their combinations, as well as random start heuristic.

- Derivative-Free Bundle Method (DFBM) of non-smooth optimization;
- Extended Cutting Angle Method (ECAM) of global Lipschitz optimization;
- Dynamical System - based Optimization (DSO) - a method of global optimization.

2.1 Non-Smooth Local Optimization

There are many practically relevant mathematical models that involve non-smooth functions, i.e. continuous functions that have a discontinuous gradient. Within the broad class of non-smooth functions, the set of locally Lipschitz-continuous functions, and in particular the class of convex functions, is of special interest. The notion of subdifferential (see [40]) is a generalization of the gradient for non-smooth convex functions. Different approaches to the generalization of this notion have been proposed subsequently: the Clarke subdifferential (see [12]) and the quasidifferential (see [13–15]) are the most important among these from the numerical point of view.

In the optimization methods based on descent, an essential step is estimating the direction of descent using some information about the subdifferential. In the DFBM method, implemented in `GANSO` , we use a special finite difference approximation to the subdifferential, called the discrete gradient [2].

The DFBM method is derivative-free, which means that it only uses the values of the objective function, but not its derivative (or its generalization) in explicit form. In essence the implemented algorithm iterates between two steps: calculation of the descent direction from the approximation to the subdifferential, and a line search along this direction.

While the DFBM is a local method (i.e., it converges to a locally optimal solution, from any starting point x), the fact that it uses an approximation to the subdifferential, allows it to converge to a sufficiently "deep" local minimum in multiextremal problems, i.e., "skip" through many annoying shallow local minima [9]. This is an important advantage of this method over other competing approaches that converge to the nearest local minimum.

2.2 Random start

In many practical problems the objective function $f(x)$ possesses many (sometimes myriads of) local minima. The goal is to locate the global minimum.

Random start is a very simple approach which involves execution of any local optimization method from a large number of "starting" points. The starting points are chosen in a random way, so that they cover the whole feasible domain. The smallest local minimum is selected as a substitute for the global minimum. There is no guarantee of the quality of the solution, but in many applications this approach delivers good results.

In `GANSO` we use Sobol quasirandom sequence of starting points. We apply the DFBM from each of these starting points. The algorithm returns the best local minimizer found in this way. The combination with DFBM gives an advantage of systematically converging to deeper local minima than with other local methods [9].

2.3 Extended Cutting Angle Method (ECAM)

Under Lipschitz continuity assumption, it is possible to estimate the smallest possible minimum of the objective function, from its recorded values at various points. It follows from the Lipschitz condition that

$$f(x) \geq \max_{k=1,\dots,K} f(x^k) - Ld(x^k, x) =: H(x), \tag{2}$$

where x^k are the points with the recorded values of $f(x^k)$, and L is the Lipschitz constant of f. The expression on the right is called the (saw-tooth) underestimate of f. By using a large number of points x^k, it is possible to approximate f closely by its underestimate H, and then use the global minimum of the underestimate to approximate that of f.

It turns out that minimizing the underestimate H is a structured optimization problem, and all its local (and the hence global) minimizers can be found explicitly. This is the basis of the ECAM [1, 4, 6–8, 41]. It uses a computationally efficient representation of local minimizers of H in a tree data structure, and computes the global minimum of f from this information. The method guarantees the globally optimal solution.

It should be noted that this method requires a very large number of function values even in problems with moderate dimension ($n \leq 5$). The issue here is not the computational algorithm, but the very fact that the points in n-dimensional space are very sparse. To build an accurate underestimate of f, the points should cover the feasible domain densely. What ECAM does however, is choose the points x^k adaptively, only in the "promising" regions, and thus improves the accuracy in the regions near deep local minimizers. ECAM employs the technique of fathoming, similar to branch-and-bound algorithms.

At the end of its execution, ECAM algorithm calls local search (DFBM) a specified number of times to improve the solution. DFBM uses the starting points supplied by ECAM.

2.4 Dynamical Systems - Based Optimization

The idea of this method (DSO) is to build a dynamical system using a number of values of the objective function, and associating with these data certain "forces". The evolution of such a system yields a globalized descent trajectory, which leads to lower values of the objective function. The DSO method typically starts with a box domain, samples the objective function within this search domain, and chooses a number of these values to define the system evolution rules. The algorithm continues sampling the domain until it converges to a stationary point [24–26].

2.5 Combination of methods

As we mentioned, in practice none of the generic global optimization algorithms alone is capable to find the global minimum of all continuous (or even Lipschitz) functions,[1] although many methods do exhibit nice theoretical convergence properties. It is a usual practice to combine two or more techniques to take advantage of combined power of several methods. Examples of this approach are numerous, e.g., multistart local search, or running local

[1]Take for example the function $f(x) = \min\{1, M||x-a||\}$, where $a \in R^n$ is some vector, and $M > 0$ is the Lipschitz constant of f. Systematic exploration of any box domain requires an astronomical number of function evaluations even for a moderate dimension n, while the success of stochastic methods and heuristics depends on the luck of choosing x in the neighborhood a (in the ball of radius $\frac{1}{M}$).

search at the final steps of a global optimization method. Below we detail some of the combinations implemented in `GANSO` library.

2.5.1 DFBM + ECAM

This combination of the local DFBM method with a special version of ECAM is designed to improve line search used in DFBM, as well as to facilitate leaving shallow local minima [5]. It works as follows.

DFBM is started with a suitable initial point and runs until it converges, i.e., it does not find any direction of descent from a point x^k. Then the algorithm chooses a specified number of promising ascent directions, and defines a bounded domain in the affine linear subspace of R^n, in which the global search is performed, typically an m-dimensional simplex. Using an estimate of the Lipschitz constant of f, the algorithm performs global search in this low-dimensional part of the domain using ECAM. ECAM runs a specified number of iterations, and if successful, finds a point with a smaller value of the objective function.

The dimensionality of the subdomain $m < n$, and there is no guarantee that global search will help escape the current local minimum. There is a trade-off between the effectiveness of this method and its numerical efficiency (as global search is numerically expensive).

We note that the line search used in DFBM can be completely substituted with the global search by ECAM in the m-dimensional simplex, as for $m < 5$ ECAM is numerically very efficient.

2.5.2 ECAM + DFBM

This is a combination of ECAM with local search [5]. At every iteration of ECAM, instead of calculating the value of the objective function $f(x)$, the algorithm executes the DFBM local optimization routine, thus taking a local minimum to which the DFBM converges when x is the starting point. Thus the objective function is replaced with an auxiliary underestimate $\hat{f}(x)$, such that all local minima of f and $\hat{f}$ coincide. ECAM searches for the global minimum of $\hat{f}$, which will also be the global minimum of f.

However $\hat{f}$ is discontinuous. It is necessary to ensure that ECAM, which is applicable to Lipschitz functions, will not loose its global convergence properties if applied to an objective function from a wrong class. This may happen, for instance, when ECAM fathoms parts of feasible domain.

We have demonstrated in [10] that the underestimate H in (2), constructed using the values $\hat{f}(x^k)$ rather than $f(x^k)$ will remain an underestimate of f, and thus no global minimizer of f can be fathomed. Thus convergence to the global minimum is preserved. On the other hand, this combination allows one to reduce the number of iterations of ECAM, required to locate the global minimum (although not to confirm it), as we have shown in [10] on a challenging molecular conformation problem.

2.5.3 ECAM + DSO

In this combination ECAM performs global search and builds a crude model of the objective function. It runs a relatively small number of iterations, and builds an underestimate H in (2). This underestimate is used to select a number of "promising" regions to thoroughly sample with another method, in this case DSO.

The algorithm selects a number of box domains based on the model H in (2), and then calls DSO method for each of these boxes. DSO performs local search within each box, and returns the best solution, which is subsequently improved by using local search DFBM.

2.5.4 Iterative DSO

This combination consists of using DSO method in a number of stages, each time reducing the search domain by some factor, using the optimal solution found in a previous stage. The natural search domain for DSO method is a box, which initially comprises the whole feasible domain. Once the first stage of DSO has converged to some solution, a smaller box around this minimizer is chosen as the search domain for the next stage. This way DSO algorithm tries to improve its results by concentrating search in a smaller neighbourhood of the current optimal solution.

An approximation to a globally optimal solution provided by a global method is usually improved by a local search (DFBM) at the final stage.

2.6 Constraints

`GANSO` allows one to specify linear equality and inequality constraints on the feasible domain D, as

$$D = \{x \in R^n : Ax = b, Cx \leq d\},$$

where A, C are matrices with n columns and m_A and m_C rows, and b and d are vectors with m_A and m_C elements respectively. It is assumed that the rows of matrices A and C are linearly independent.

`GANSO` performs preprocessing of the constraints. The inequality constraints (except box constraints) are transformed into equality constraints with the help of m_C non-negative slack variables. Then the feasible domain is expressed as

$$D = \{y \in R^{n+m_C} : \tilde{A}y = \tilde{b}\}.$$

Following, $n + m_C - m_A$ independent variables are identified (basic variables y_B). The rest of the variables are expressed as an affine combination of the basic variables $y_{NB} = Gy_B + g$, where the matrix G and vector g are obtained from the original parameters A, b, C, d using linear algebra operations. The choice of the basic variables is governed by numerical stability of the matrix inversion operation when computing G and g (we use LU factorization with pivoting).

Box constraints on the basic variables are dealt with by the algorithms directly (they are natural in global optimization, whereas the DFBM adapts its approximation to the subdifferential), while box constraints on the non-basic variables are dealt with using an exact penalty function, calculated internally). All these procedures are transparent to the user, who only needs to provide the arrays containing the elements of A, b, C, d to the algorithm.

Nonlinear equality and inequality constraints are generally dealt with using exact penalty functions. The penalty parameters depend very much on the type of the constraints, and should be implemented by the user. It involves modifying the value of the objective function by using an additive penalty term, such as

$$objf = f(x) + P||c(x)_+||,$$

where P is the penalty parameter, $c(x)$ denotes the vector of constraint violations, and c_+ denotes its positive part (i.e., its i-th component is not zero only when the i-th constraint is violated. The norm is arbitrary, see [11].

It should be noted that none of the optimization methods alone (or in combination) cannot guarantee the best optimal solution in a practical setting (global convergence can be proved in theory, but it requires astronomical computing time). Therefore the user should experiment with different methods in order to choose the one most suitable for her particular problem.

3 Implementation of the library

3.1 GANSO library

The described methods of optimization are implemented in the programming library GANSO in C++ language. The algorithms can be accessed via class interface or via a number of C-style procedures. These procedures allow one to call GANSO methods from other programming languages, like Fortran, as well as other packages like Matlab and Maple, as described in the following sections.

C++ class interface involves one main class called Ganso, which provides the interface to all the optimization methods. There is also a number of C-style procedures, which simply call the relevant member functions of Ganso class. The description of all the methods and parameters is beyond the scope of this paper, and it is done in GANSO manual [16]. Here we just illustrate these methods on two examples, the DFBM method and a combination of DFBM with ECAM.

```
class Ganso {
 public:
// various optimization methods
    int MinimizeDFBM(int dim, double* x0, double *val, USER_FUNCTION f,
        int lineq, int linineq, double* AE, double* AI, double* RHSE,
        double * RHSI,double* Xl, double* Xu, int* basic, int maxiter);

    int  MinimizeDFBMECAM(int dim, double* x0, double *val, USER_FUNCTION f,
        int lineq, int linineq, double* AE, double* AI, double* RHSE,
        double * RHSI, double* Xl, double* Xu, int* basic, int maxiter,
        int iterECAM, int dimECAM);
// other methods...
...
 };
```

The minimization methods require a reference to the user supplied procedure for calculation of the objective function, as well as the number of variables (dimension), matrices of constraints, upper and lower bounds on the variables and also the parameters of the algorithms, such as the maximal number of iterations. The reference to the objective function procedure is defined as

```
typedef void (*USER_FUNCTION)(int *n, double *x, double *f);
```

where the first argument is the dimension, second argument is the array of size n containing the coordinates of x, and f is the value of $f(x)$ to be calculated.

Note that most of the parameters are standard for all the methods in GANSO library, but a few are method-specific. This allows one to substitute one method for another with little programming effort. There are also a number of simplified methods, which do not require matrices of linear constraints (in cases where only box constraints are provided).

Procedural interface is useful for calling GANSO methods from languages like C and Fortran, and also from packages like Matlab and Maple. These C-type functions have exactly the same names and lists of arguments as their Ganso class counterparts, and in fact are nothing but wrappers. This means that the user has two options: to declare an instance of the class Ganso and call its member functions, or to call the C procedures with the same name. Class interface provides more flexibility though, as the user can access member variables and call various member functions without deleting an instance of the class.

The following example illustrates the use of GANSO by a C++ program.

```
 int main() {
    ...
// declare an instance of Ganso class
    Ganso G;
// call its members
    int retcode=G.MinimizeECAM(2,x0,&val,myfunction,0,0,40,
        NULL,NULL,NULL,NULL, boundsL, boundsU, NULL,3000);

    retcode=G.MinimizeECAMDFBM_0(2,x0,&val,myfunction,40,
       boundsL, boundsU,100);

// call C functions
    retcode = MinimizeECAM(2,x0,&val,myfunction,0,0,40,
        NULL,NULL,NULL,NULL, boundsL, boundsU, NULL,3000,10);

    retcode=MinimizeECAMDFBM_0(2,x0,&val,myfunction,40,
       boundsL, boundsU,100);
    return 0;
}
```

GANSO provides all its functions in a special syntax suitable for calling from Fortran. They mimic all the methods of Ganso class (the description of parameters is almost identical), and differ only in how they pass parameters.

3.2 Interfaces with Matlab and Maple

Many scientists use such systems as Matlab, Maple and Mathematica to do their routine calculations. These systems provide only a few basic nonlinear optimization methods [27,28,34,36,39,44], although there are additional toolboxes that implement more advanced methods of global optimization (e.g., LGO [34,35]). Availability of novel optimization methods as subroutines in the mentioned systems is important for their wider dissemination. However implementation of external libraries is not that straightforward. In this section we describe our experiences with Matlab and Maple interfaces.

3.2.1 Matlab interface

Matlab is a widely used package, especially in engineering community. It has a number of internal local optimization routines, such as basic `fminsearch`, and more advanced methods available in the Optimization toolbox [28].

Matlab implements mechanisms for calling external functions from dynamically linked libraries (dlls), as if they were Matlab's internal functions. Thus users see no difference between calling Matlab's own function `fminsearch()` and an external function `minimizeDFBM()`, except for the list of parameters. However, a proper interface layer should be implemented by the developer of such a library. The interface consists of a special wrapper dll, which does the conversion of the parameters passed from Matlab to another library, such as `GANSO` . For instance, conversion of matrices from Fortran column-major convention adopted in Matlab, to C row-major convention, used in `GANSO` .

However the most important issue is the interface to the user defined objective function. All optimization methods make calls to the objective function, which is supplied as a Matlab procedure. On the other hand, libraries such as `GANSO` require a C-type procedure. The wrapper library provides an interface between `GANSO` and Matlab, through which `GANSO` can call Matlab procedures.

Because of potentially large number of function evaluations, this wrapper must be very efficient. In our implementation, we used a number of programming techniques to achieve maximum efficiency. Matlab's internal `mexCallMATLAB()` procedure [28] provides the most efficient way of calling Matlab's functions (i.e., the user defined objective function). In Matlab's current implementation this procedure takes on average 2.1×10^{-5} sec to execute the user's objective function in Matlab [2].

Below is an example of Matlab code calling GANSO .

```
% declare the objective function in the file myfunction.m
   function f = myfunction(x)
     f=x(1)*x(1)+x(2)*x(2)-cos(18*x(1))-cos(18*x(2));

% Main Matlab procedure: define vectors of box constraints
% and other parameters
    dimension=2;
    Lo=[-1,-1];
    Up=[1,1];
    iter=1000;
    LipConst=10.0;
    x0=[1,1];
% execute Ganso global ECAM method
[val,x0]=minimizeGanso_0('ECAM',dimension,x0,@myfunction,LipConst,Lo,Up,iter);
    x0 % Print the solution
    val  % Print the function value at the solution
% execute local search (DFBM) from a staring vector(0.5,0.5)
    x0=[0.5,0.5];
[val,x0]=minimizeGanso_0('DFBM',dimension,x0,@myfunction,Lo,Up,iter);
    x0 % Print the solution
    val  % Print the function value at the solution
```

Our experiments with Matlab show that for global optimization in a few variables ($n < 8$) and local non-smooth optimization (for $n < 30$), Matlab interface to GANSO is an efficient user friendly tool. However for higher dimension, because of a much larger required number of objective function evaluations, it does not compete with calling GANSO from a C or Fortran code.

3.2.2 Maple interface

Maple is another popular scientific package, mostly known for its symbolic processing capabilities [27]. It also implements a number of classical numerical methods, but its numerical nonlinear optimization methods are very limited. There are external commercially available optimization toolboxes (e.g., LGO [34, 35]) which provide access to some modern global optimization methods.

[2]On a Pentium M, 2 GHz processor under MS Windows, Matlab version 7.1.

Maple allows one to call C-type subroutines from external dlls directly using a specific syntaxis, however this method is not suitable when callback functions are used (calls to the user's objective function written in Maple's language). The alternative approach to implementing the interface to `GANSO` methods consisted in using a wrapper dll, which does conversion of parameters (from Maple's column-major to C row major convention) and provides a wrapper for a callback function, similar to Matlab.

Here again, calls from `GANSO` to Maple's functions are implemented internally in Maple, and the efficiency of such calls is crucial. Unlike Matlab, Maple's developers implemented two alternative ways for doing this (via functions `EvalMapleProc()` and `EvalhfMapleProc()`) [27]. The first function is similar to Matlab's `mexCallMATLAB()`, i.e., it provides a generic way to execute any Maple command. It is general, but numerically inefficient (average call time was 5.0×10^{-5} sec on Pentium M, 2 GHz processor).

The second method uses "hardware floats", and is very efficient (average call time was 3.1×10^{-7} sec on the same processor). However the drawback is its limited versatility. All parameters to Maple's function must be explicit, no vector notation is allowed, and only a few simplest Maple commands can be used. For example, to calculate the norm of a vector in 5d one should define the Maple function as

```
f := proc(x1,x2,x3,x4,x5)
  sqrt(x1^2+x2^2+x3^2+x4^2+x5^2);
end proc;
```

Thus, for small scale optimization problems, the generic Maple syntaxis is suitable, as it allows one to use most Maple functions and construction (such as loops). For larger scale problems, we implemented the alternative numerically efficient way of evaluating the objective function, but at the expense of generality and expressivity. The following example illustrates the use of `GANSO` in Maple.

```
# definition of an external subrotine found in Ganso
 MindfbmOHF := define_external('MWRAP_HFMinimizeDFBM_0','MAPLE',
    LIB="mwrap_ganso.dll"):

 f := proc( x1,x2,x3 ) #user's objective function
   x1^2+(x2-1)^2+x3^2;
 end proc;
```

```
# defining vectors of constraints
    dimension:=3;
    Lo:=Vector(1..3,datatype=float[8],[-1,-1,-1]):
    Up:=Vector(1..3,datatype=float[8],[2,2,2]):
    iter:=1000: val:=1.0:
# starting point
    x0:=Vector(1..3,datatype=float[8],[1,1,1]);

# executing local search (DFBM method)
    Mindfbm0HF(dimension,x0,val,f,Lo,Up,iter);
    print(x0);
```

We end this section by emphasizing that the most numerically efficient way of using optimization libraries is to implement the objective function in C, Fortran, or other "traditional" programming languages. Calling external optimization methods from other packages involves unavoidable overheads, which limit the speed of execution. On the other hand, these packages provide for many users familiar and friendly environment, and also contain their "legacy" code. We realize that many users need flexibility for some experimentation with their optimization problems, and in this context Matlab and Maple interfaces to `GANSO` library provide a suitable access to a number of modern global and non-smooth optimization methods.

4 Conclusions

The success of any generic optimization method ultimately depends on its wide acceptance by the users. The acceptance is influenced by a number of factors. The main one is, of course, the ability to solve specified optimization problems. However, given that a general non-convex global optimization problem is mathematically unsolvable, this criterion cannot be fulfilled by any generic method.

Then additional factors, such as availability, easiness of implementation and usage, interfaces with popular packages, speed of execution, documentation and examples, etc., play the role. Various heuristics have gained popularity not so much due to the quality of reported solutions, but to other mentioned factors. Given that optimization methods based on advanced mathematical theories (such as abstract convexity, non-smooth analysis) require a significant effort to implement them, availability of user-friendly programming libraries is very important.

Many practitioners use integrated packages such as Matlab, Maple and Mathematica. Provision of a suitable programming interface, so that optimization methods could be executed from these packages with little effort is also very important.

In this paper we reported on implementation of a number of advanced global and non-smooth optimization methods that have been recently developed based on fundamental results in abstract convex and non-smooth analysis. We presented the main ideas behind each of the implemented methods, as well as combined methods and treatment of simple constraints. We also outlined the key implementation issues, such as the programming interface with C/C++ and Fortran languages, as well as integrated systems Matlab and Maple. Wide dissemination of modern optimization methods is impossible without addressing these implementation issues.

Our experiences with developing interfaces with Matlab and Maple packages have been generally positive, and we plan to extend our work to other packages, such as Mathematica and R. The `GANSO` library is available for download from [16].