

Programming library
GANSO
Global And Non-Smooth Optimization
Version 1.0
User Manual

Centre for Informatics and Applied Optimisation
School of Information Technology and Mathematical Sciences
University of Ballarat
Ballarat, Victoria, Australia
www.ballarat.edu.au/ciao

Copyright CIAO, 2005.

Contents

1	Introduction	5
2	Installation instructions and Licensing	7
2.1	Library contents	7
2.2	License agreement	7
2.3	Installation: Windows	9
2.4	Installation: Unix/Linux	9
2.5	Linking against GANSO	9
2.5.1	Windows: MS Visual C++	10
2.5.2	Windows: G++	10
2.5.3	Unix/linux	10
2.5.4	Calling GANSO from other packages	11
3	Optimization methods	13
3.1	Non-Smooth Local Optimization	14
3.2	Global Optimization	15
3.2.1	Random start	15
3.2.2	Extended Cutting Angle Method (ECAM)	16
3.2.3	Dynamical Systems - Based Optimization	17
3.3	Combination of methods	17
3.4	Constraints	19
4	Description of the library	21
4.1	Programming interface	21
4.2	Members of Ganso class	22
4.3	Error codes	25
4.4	Statistics	26
4.5	Procedural interface	27

4.6	Fortran interface	27
4.6.1	g77 compiler	29
4.6.2	lf95 compiler	32
5	Examples of usage	35
5.1	Sample code	35
5.2	Linking	40
5.3	Tips	40
5.4	Where to get help	41

Chapter 1

Introduction

This manual describes the programming library **GANSO**, which stands for **Global And Non-Smooth Optimization**, which implements several methods of optimization.

Specifically, **GANSO** implements the following methods

- Derivative-Free Bundle Method (DFBM) of nonsmooth optimization;
- Extended Cutting Angle Method (ECAM) of global Lipschitz optimization;
- Dynamical System – based Optimization (DSO) - a method of global optimization;
- Random Start local optimization;
- Various combinations of the above methods.

A detailed description of these methods is given in a series of papers listed in the bibliography section. This manual is programmer-oriented, it describes the usage of the library **GANSO** in applications, including compilation, linking against **GANSO**, description of the methods and procedures and their parameters. The manual also provides a number of example programs.

Chapter 2 describes the installation instructions. Chapter 3 gives a brief description of optimization methods used in this library. The description of the programming library **GANSO** is given in Chapter 4. Examples of its usage are provided in Chapter 5.

Chapter 2

Installation instructions and Licensing

2.1 Library contents

GANSO is distributed under the license from CIAO, described in the sequel, which allows the users unlimited use of the library in their software, as long as all other conditions of the License agreement are met.

The library is distributed in compiled form (as binary library files), C header files, example programs in C and Fortran and this user manual. Installation involves copying the contents of the library into a designated directory.

Before using the software and its technical documentation, please take a few moments to read the following legal and contact information.

2.2 License agreement

The **GANSO** software product and its entire documentation are developed and maintained by the Centre for Informatics and Applied Optimisation, School of Information Technology and Mathematical Sciences, University of Ballarat, Australia. The developer will be abbreviated below as CIAO.

The **GANSO** library – including all files distributed as part of the product delivery – and its documentation may not be changed or modified in any way, except by CIAO, or following the written permission of CIAO. All proposed changes should be requested by contacting CIAO.

Without a proper license issued to users individually by CIAO, no part of the **GANSO** documentation and/or of the software may be stored, reproduced, transmitted, transferred, or used in any form or by any means, by any information storage and retrieval system or process.

GANSO is provided to registered users in compiled (static or dynamic link library) form. It is specifically prohibited to apply reverse engineering or any other form of internal program structure analysis to the **GANSO** software product.

Registered **GANSO** users are granted permission to use all information summarized by the User Manual, and to apply all **GANSO** software components and test examples in their own work, without any restriction. A reference to the software and its documentation will be appreciated in published work in which **GANSO** is used.

If **GANSO** is used as part of a new software application, then it is requested to maintain all **GANSO** copyright and licensee information in that application, including all reports generated by the **GANSO** software.

Regardless of how a copy of **GANSO** is obtained, it is requested that all users comply with the licensing and registration provisions if they continue to use the software.

Limited Warranty and Disclaimer

CIAO guarantees that all shipped **GANSO** products are free from defects in materials and workmanship, under normal use and service for a period of 90 days.

CIAO reserves the right to revise the **GANSO** software and its documentation, with no obligations to notify any person or organization of such revision. Information updates will be provided upon request.

The **GANSO** library is distributed 'as is'. CIAO and library developers specifically disclaim all warranties – express or implied – related to the merchantability and fitness of the **GANSO** software for a particular purpose or application use.

In no event shall CIAO and its developers be liable for loss of profit, commercial, business related, or any other damage-including, but not limited to special, incidental, consequential, or any other explicit and implied damages—as a consequence of using, or inability to use, the **GANSO** library.

Technical Support

Registered users of **GANSO** are entitled to limited technical support, provided by CIAO. Please note that e-mail is the preferred way of communica-

tion, except in cases of extreme urgency. Consulting fees may be charged for direct assistance via telephone/fax.

Present and prospective **GANSO** users are encouraged to contact CIAO, should they wish to discuss specific applicability issues, and possibilities of research and/or commercial co-operation. All constructive suggestions and comments that could lead to improvements of **GANSO**, its documentation, and its applicability are welcome and appreciated. We are much interested to hear user comments and suggestions; test models and application examples are also welcome.

Thank you for your interest in our software.

2.3 Installation: Windows

GANSO library is distributed in the form of a dynamically linked library (DLL), together with the corresponding LIB file, header files and the examples of usage. All these files are archived into a single file `ganso1.zip`. The installation involves extracting the contents of this file into a designated directory using utility program such as WinZip.

2.4 Installation: Unix/Linux

GANSO library is distributed in the form of a static library (`libganso.a`), together with the header files and the examples of usage. All these files are archived into a single file `ganso1.0.tar.gz`. The installation involves extracting the contents of this file into a designated directory using utility programs such as `gunzip` and `tar`.

```
gunzip ganso1.0.tar.gz
tar -xvf ganso1.0.tar
```

2.5 Linking against GANSO

The package contains a number of header files, and binary static library files (we also provide `.dll` files for windows developers). To link against **GANSO** libraries, add

```
#include "ganso.h"
```

line to your code, and link against `ganso` library. Below we provide the details of how to link against `ganso` using different compilers.

2.5.1 Windows: MS Visual C++

GANSO is distributed as a DLL file and the corresponding LIB file called `gansodll.dll` and `gansodll.lib`. In your project settings, choose LINK option and add `gansodll.lib`. Do not forget to include `gansodll.dll` library with your executable (it should reside in your program's directory or be on the path). The DLL is compiled using `__stdcall` option, for compatibility with other software, therefore you should use `__stdcall` option as well (in Project Settings – C/C++ – Code generation, select Calling convention).

2.5.2 Windows: G++

When using `gcc` or `g++` compilers on Windows, one normally uses a unix-like shell like MinGW. The compilation/linking instructions are analogous to unix/linux. GANSO is distributed as a static library called `libganso.a`. Linking is done by including this file with your other object files. See the next section, and also the makefile included with the examples.

2.5.3 Unix/linux

On unix/linux platform, in case you do not have administrator privileges, you can install GANSO into your home directory. In this case linking is done by specifying the full path, like

```
g++ -o example1 example1.o -L$(LIB_PATH) -lganso -lm
```

or using `libganso.a` without `-l` switch, e.g.,

```
g++ -o example1 example1.o $(LIB_PATH)libganso.a -lm
```

See also example makefiles in the `examples` directory. A number of sample C++ source and makefile files are provided in this package. To test the library change to `examples` directory and compile and run the examples using

```
make
./example1
./example2
./example2c
```

On some unix machines the path to standard libraries, like `libstc++.so` and `libg2c.so`, is not set by default. Usually they reside in `/usr/local/lib`, and this path might need to be added (e.g., the variable `LD_LIBRARY_PATH` set and exported).

2.5.4 Calling GANSO from other packages

GANSO library can also be called from other programs, such as Mathematica, Matlab, and Maple. Please contact CIAO if you need specific instructions.

Chapter 3

Optimization methods

GANSO library implements a number of methods of Global and Non-Smooth Optimization methods developed by CIAO and its associates. These methods aim at solving the following generic optimization problem

$$\begin{aligned} & \text{minimize } f(x) \\ & \text{subject to } x \in D \subset R^n. \end{aligned} \tag{3.1}$$

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

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

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). The Lipschitz continuity is expressed as

$$|f(x) - f(y)| \leq Ld(x, y),$$

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. Computation

of the direction of descent at those points where f is not differentiable is also tricky. Familiar methods, such as quasi-Newton, or conjugate gradient, will simply not work on 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., [BGLS00, HT93, HPT00, HP95, MW93, Pin96, SS00]. The user is encouraged to familiarize him/herself with some of these books, to appreciate the difficulty of the problem and the unavoidable limitations of any generic method for its numerical solution.

GANSO implements four different approaches to global non-smooth optimization, and also their combinations. For unconstrained local non-smooth optimization we use the Derivative Free Bundle Method (DFBM), based on finite difference approximation to the subgradient [Bag03, Bag02]. For global optimization (necessarily on some compact domain D) we use simple Random start method, the Extended Cutting Angle method (ECAM) [Rub00, BR01, BB02, Bel04, Bel05], and a method based on trajectories of Dynamical Systems (DSO). Some combinations of these methods are also implemented. The idea is that methods that use different approaches may enhance each other in some way.

It should be noted that none of the methods alone (or in combination) cannot guarantee the best optimal solution in 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.

3.1 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 [Roc70]) 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 [Cla83]) and the quasisubdifferential (see [DR86, DR95]) 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 [Bag02].

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. This is an advantage of this method over other competing approaches that converge to the nearest local minimum.

3.2 Global Optimization

In many practical problems the objective function $f(x)$ possesses many (sometimes myriads of) local minima. The goal is to locate the global minimum (see Fig.3.1).

3.2.1 Random start

This is a very simple approach which involves using 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 on the quality of the solution, but in many applications this approach delivers good results.

In `GANSO` we use the 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.

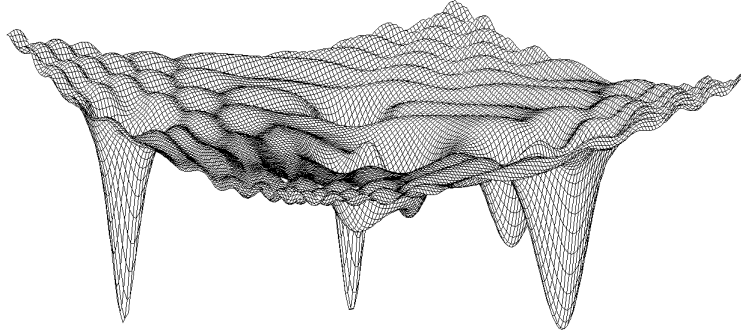


Figure 3.1: Graph of a typical multiextremal function, with a large number of shallow local minima.

3.2.2 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

$$f(x) \geq \max_{k=1,\dots,K} f(x^k) - Ld(x^k, x) =: H(x), \quad (3.2)$$

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 hence global) minimizers can be found explicitly. This is the basis of the ECAM [Rub00, BB02, Bel04, Bel05]. 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 choosing the points x^k adaptively, only in the "promising" regions, and thus improving 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.

3.2.3 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 [Mam04, MO05, MRY05].

3.3 Combination of methods

DFBM + ECAM

This is a combination of the DFBM with a version of ECAM, designed to improve line search used in DFBM, as well as to facilitate leaving shallow local minima. If at some stage DFBM algorithm does not find a direction of descent, then it chooses the specified number of promising ascent directions, and spans a linear subspace of R^n , in which global search is performed. Using the boundaries of the feasible domain, and its estimate of the Lipschitz constant of f , DFBM algorithm sets a search domain for ECAM in this

subspace. ECAM performs a specified number of iterations, and if successful, finds a point with a smaller value of the objective function.

The dimension of the subspace is typically smaller than n , and therefore there is no guarantee that global search will 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).

ECAM + DFBM

This is a combination of ECAM with local search. We use the global algorithm ECAM at the outer level, and the DFBM at the inner level. That is, 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 taking x as the starting point. From the point of view of DFBM, ECAM serves as a driver to generate appropriate starting points for local optimization.

While from the point of view of ECAM the objective function is no longer continuous, the underestimate H in (3.2) will remain an underestimate, and the algorithm will converge to the global minimum of f .

ECAM + DSO

In this combination ECAM performs global search and builds a crude model of the objective function. It then calls DSO a specified number of times, and provides it with box domains, chosen according to the model. DSO performs its search within the supplied box domains.

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 point 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.

3.4 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 artificial non-negative variables (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 global optimization, 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 [BGLS00].

The nonlinear constraints should be implemented by the user, and supplied to the algorithms through the values of the modified objective function. The GANSO algorithms will treat the nonlinearly constrained and unconstrained problems equally, and it is the users responsibility to choose the right penalty parameters and to check the feasibility of the solution returned by GANSO .

Chapter 4

Description of the library

4.1 Programming interface

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.

There is one main class which provides the interface to the optimization methods implemented in **GANSO**, called **Ganso**. There is also a number of C style procedures, which simply call the relevant member functions of **Ganso** class. They facilitate calling **GANSO** methods from such languages like Fortran, or packages like Mathematica or Matlab.

```
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...
...
};
```

4.2 Members of Ganso class

Class **Ganso** implements a number of global and non-smooth optimization methods and their combinations described in Chapter 3. Typically these methods require specifying a reference to the user supplied procedure for calculation of the objective function, the number of variables, matrices of constraints, 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 returned.

```
int MinimizeDFBM(int dim, double* x0, double *val, USER_FUNCTION f, int lineq,
int llineq, double* AE, double* AI, double* RHSE, double * RHSI, double*
Xl, double* Xu, int* basic, int maxiter)
```

Performs local nonsmooth optimization using DFBS. Parameters:

dim- is the number of variables (the dimension of the problem),

x0- is the starting point (initial approximation) on entry, and the final locally optimal solution on exit,

val- is the value of the objective function at the final point,

f - is the reference to the user supplied function,

lineq - is the number of linear equality constraints,

llineq - is the number of linear inequality constraints.

The remaining parameters are optional (default values are NULL).

AE is the array of size $lineq \times dim$ containing the entries of the matrix of linear equality constraints (C row-major indexing: $AE[i * dim + j]$ contains AE_{ij}),

AI is the array of size $llineq \times dim$ containing the entries of the matrix of linear inequality constraints (same indexing as *AE*),

RHSE is vector of size *lineq* of the right hand side of the system of linear equality constraints,

RHSI is vector of size *llineq* of the right hand side of the system of linear inequality constraints,

Xl is the vector of size *dim* of the lower bounds on the variables, NULL if no lower bounds,

Xu is the vector of size *dim* of the upper bounds on the variables, NULL if no upper bounds

basic is an optional vector of integers of size *dim*, specifying the indices (0-based) of the desired basic variables, if NULL then chosen automatically,

maxiter is the upper limit on the number of iterations (the default is 10^4).

This method returns 0 when successful, otherwise the returned value is the error code, as described later.

Unless otherwise specified, the parameters in the subsequent methods have the same meaning as in `MinimizeDFBM`.

```
int MinimizeRandomStart(int dim, double* x0, double *val, USER_FUNCTION f, int
lineq, int linlineq, double* AE, double* AI, double* RHSE, double * RHSI,
double* Xl, double* Xu, int* basic, int maxiter)
```

Performs a series of local nonsmooth optimizations from random starting points within a specified box domain using DFBM. Parameters: same as in `MinimizeDFBM`, except:

maxiter is the number of random starts. The number of iterations of DFBM local search is limited to 1000. The final optimization is performed with the limit of 10000 iterations.

```
int MinimizeDFBMECAM(int dim, double* x0, double *val, USER_FUNCTION f, int
lineq, int linlineq, double* AE, double* AI, double* RHSE, double * RHSI,
double* Xl, double* Xu, int* basic, int maxiter, int iterECAM, int dimECAM)
```

Performs local nonsmooth optimization using DFBM, in combination with ECAM to improve line search and escape local minima. Parameters: same as in `MinimizeDFBM`, except:

iterECAM is the number of iterations of ECAM.

dimECAM is the dimension of the subspace on which ECAM performs global search, should be smaller than *dim* and typically smaller than 20.

```
int MinimizeECAM(int dim, double* x0, double *val, USER_FUNCTION f, int lineq,
int linlineq, double LC, double* AE, double* AI, double* RHSE, double
* RHSI, double* Xl, double* Xu, int* basic, int maxiter, int iterLocal)
```

Performs global Lipschitz optimization in the domain specified by the box constraints *Xl*, *Xu* (should not be NULL). Parameters: same as in `MinimizeDFBM`, except:

LC – the estimated Lipschitz constant of the objective function. It can be overestimated at the expense of the running time, underestimation may lead to premature termination of the algorithm. In this case the returned value is -2 .

iterLocal – the number of times local optimization method DFBM is called from the starting points chosen by ECAM, at the end of its execution. The default value is 1.

```
int MinimizeECAMDFBM(int dim, double* x0, double *val, USER_FUNCTION f, int
lineq, int linlineq, double LC, double* AE, double* AI, double* RHSE,
double * RHSI, double* Xl, double* Xu, int* basic, int maxiter)
```

Performs global Lipschitz optimization in the domain specified by the box constraints Xl , Xu (should not be NULL). Every function value is a local minimum found by DFBM. Parameters are identical to those of `MinimizeECAM`.

```
int MinimizeDSO(int dim, double* x0, double *val, USER_FUNCTION f, int lineq,
int llineq, double LC, double* AE, double* AI, double* RHSE, double
* RHSI, double* Xl, double* Xu, int* basic, double penalty, int speed,
int precision)
```

Performs global optimization in the domain specified by the box constraints Xl , Xu (should not be NULL), using the dynamical systems approach. Parameters the same as in `MinimizeDFBM`, except:

penalty – penalty parameter when inequality constraints are specified. Used to penalize constraints violations.

speed – is the speed factor, controlling how exhaustively the box domain is explored, it should take values from 1 to 4, 1 is the fastest but less accurate, 4 is slower but has a better chance to find the global minimum.

precision – is the desired precision, related to number of steps in the evolution of the dynamical system. This parameter can have values from 1 to 9 (9 means better accuracy but it is slower. 3 is the default).

```
int MinimizeIterativeDSO(int dim, double* x0, double *val, USER_FUNCTION f,
int lineq, int llineq, double LC, double* AE, double* AI, double* RHSE,
double * RHSI, double* Xl, double* Xu, int* basic, double penalty, int
speed, int precision, int rounds)
```

Performs iterative global optimization in the domain specified by the box constraints Xl , Xu (should not be NULL), using the dynamical systems approach. Parameters the same as in `MinimizeDSO`, additional parameter

rounds – is the number of rounds in the iterative process, can range from 1 to 10.

```
int MinimizeECAMDSO(int dim, double* x0, double *val, USER_FUNCTION f, int lineq,
int llineq, double LC, double* AE, double* AI, double* RHSE, double
* RHSI, double* Xl, double* Xu, int* basic, int maxiter, int iterDSO)
```

Performs global Lipschitz optimization in the domain specified by the box constraints Xl , Xu (should not be NULL). Every function value is a local minimum found by DSO algorithm. Parameters are identical to those of `MinimizeECAM`.

Parameter *iterDSO* refers to the number of times DSO is called in the box regions supplied by ECAM.

Simplified version of the above methods for problems with no linear equality/inequality constraints: the list of arguments does not include these constraints (nor penalty parameter), the remaining parameters have the same as earlier.

```

int MinimizeDFBM_0(int dim, double* x0, double *val, USER_FUNCTION f, double*
    Xl, double* Xu, int maxiter)
int MinimizeDFBMECAM_0(int dim, double* x0, double *val, USER_FUNCTION f, double*
    Xl, double* Xu, int maxiter, int iterECAM , int dimECAM)
int MinimizeRandomStart_0(int dim, double* x0, double *val, USER_FUNCTION f,
    double* Xl, double* Xu, int maxiter)
int MinimizeECAM_0(int dim, double* x0, double *val, USER_FUNCTION f, double
    LC, double* Xl, double* Xu, int maxiter, int iterLocal)
int MinimizeECAMDFBM_0(int dim, double* x0, double *val, USER_FUNCTION f, double
    LC, double* Xl, double* Xu, int maxiter)
int MinimizeDSO_0(int dim, double* x0, double *val, USER_FUNCTION f, double*
    Xl, double* Xu, int speed, int precision)
int MinimizeIterativeDSO_0(int dim, double* x0, double *val, USER_FUNCTION f,
    double* Xl, double* Xu, int speed, int precision, int rounds)
int MinimizeECAMDSO_0(int dim, double* x0, double *val, USER_FUNCTION f, double
    LC, double* Xl, double* Xu, int maxiter, int iterDSO)

```

4.3 Error codes

Each method returns an integer containing the error code, which should be analyzed. Return value 0 indicates no errors (successful termination of the algorithm). Negative value means premature termination due to an error. Otherwise the errors are:

- -100 : the specified number of linear equality or inequality constraints is negative.
- -101 : $lineq > 0$ but the matrix AE or vector $RHSE$ are null.
- -102 : $linineq > 0$ but the matrix AI or vector $RHSI$ are null.
- -103 : too many inequality constraints: the number of basic variables is smaller than 1.
- -104 : no upper or lower limits provided for ECAM or DSO.
- -105 : a singular matrix of transformation of variables, most likely AE has linearly dependent rows.

- -1 : not enough memory.
- -2 : Lipschitz constant specified for ECAM is too small.
- -200 : Demo version only: too many variables or constraints.

4.4 Statistics

`int GetFunctionCalls()`

Returns the total number of function evaluations until termination of the algorithm.

`int GetFunctionCallsSolFound()`

After `MinimizeECAM` and `MinimizeECAMDFOB` only: returns the number of function evaluations until the best solution was found.

`double GetECAMLowerBound()`

After `MinimizeECAM` and `MinimizeECAMDFOB` only: returns the value of the lower bound on the objective function. It can be compared to the solution returned by these methods to evaluate its error margin.

4.5 Procedural interface

In addition to the class interface, **GANSO** provides the user with procedural interface, which is useful for calling **GANSO** methods from languages like C and Fortran, and also from packages like Mathematica 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. Internally these functions declare an instance of **Ganso** class and call its members, passing to them the full list of arguments. Their description is identical to those of **Ganso** class members.

This means that the user has two options: declare an instance of the class **Ganso** and call its member functions, or 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 destroying the class.

Example:

```
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;
}
```

4.6 Fortran interface

It is possible to call subroutines from **GANSO** library from FORTRAN. There are some programming tricks however, necessary for calling C/C++ functions from Fortran.

Firstly, by default Fortran compilers automatically add an underscore ”_” at the end of function names, this can be disabled with a compilation switch (e.g., `-fno-underscoring` in g77) compilation flag in your make file, see examples.

Secondly, Fortran passes all parameters by reference, not by value. Thirdly, linking a fortran program with C++ library sometimes requires a wrapper, which makes the main program being declared in C and not in Fortran code.

`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 (by reference only). For example the function `MinimizeECAM` unction has the following alias

```
int minimizeecam(int* dim, double* x0, double *val, USER_FUNCTION
f, int* lineq, int* llineq, double* LC, double* AE, double* AI,
double* RHSE, double * RHSI, double* Xl, double* Xu, int* basic,
int* maxiter, int *iterLocal)
```

Differences in Fortran-specific declarations

Because Fortran does not pass NULL pointers to the subroutines, it is necessary to specify explicitly whether some of the parameters are meaningful or not. The following two subroutines have an additional parameter *boxconstraints*, which tells the library whether box constraints are used (0 - no constraints, 1 - only lower bounds, 2 - only upper bound, 3 - lower and upper bound).

```
int minimizedfbm(int* dim, double* x0, double *val, USER_FUNCTION
f, int* lineq, int* llineq, double* AE, double* AI, double* RHSE,
double * RHSI, double* Xl, double* Xu, int* basic, int* maxiter,
int* boxconstraints)
```

```
int minimizedfbmECAM(int* dim, double* x0, double *val, USER_FUNCTION
f, int* lineq, int* llineq, double* AE, double* AI, double* RHSE,
double * RHSI, double* Xl, double* Xu, int* basic, int* maxiter,
int* iterECAM , int* dimECAM, int* boxconstraints)
```

Note that in all other subroutines, box constraints must be set.

The array *basic* specifies which of the variables should be treated as basic (if there are equality constraints). In C, the contents of the array is 0-based, in Fortran it is 1-based, i.e., the user should specify the entries as (1, 2, 3, ...) to set the first, second and the third variables as basic. In C, it would have been (0, 1, 2, ...).

4.6.1 g77 compiler

g77 compiler requires declaring the main program as subroutine, and having a small C wrapper, as illustrated below. It also requires `-fno-underscoring` flag.

The following is an example of a Fortran program calling `GANSO` functions. Notice using the variables and not the numbers to pass the parameters. Also notice that in Fortran code we use `subroutine` and not `program` for the main program. We also need an additional C file `wrapper.c` to properly link Fortran code.

Also note that when specifying matrices of constraints in Fortran, the matrix should be defined in the transposed form. This is because Fortran stores matrices columnwise, whereas C does this rowwise. A simple way to define the constraints in Fortran is to use one dimensional array, like in the following example

```
      double precision AE(6)
c to store the matrix 2 x 3
c First row of the matrix AE
      AE(1) = 1.
      AE(2) = 1.
      AE(3) = 1.
c Second row
      AE(3+1) =0.
      AE(3+2) =1.
      AE(3+3) =2.
```

and in general use `AE((i-1)*columns + j)` to access the element AE_{ij} .

```

        subroutine Rastrigin
c This is our main program
        double precision boundsL(2),boundsU(2),x0(2),val
        integer retcode,nvar
        double precision LC
        external func
c    Define the box constraints
        boundsL(1)=-1.
        boundsL(2)=-1.
        boundsU(1)=1.
        boundsU(2)=1.
        x0(1)=0.5
        x0(2)=0.5
c number of variables, constrains and the Lipschitz constant
        nvar=2
        neq=0
        nineq=0
        LC=40.
        Null=0
        niter=3000
        nlocal = 10
        retcode=minimizeecam(nvar,x0,val,func,neq,nineq,LC,Null,Null,
2Null,Null,boundsL,boundsU,Null,niter,nlocal)
        print *,"ECAM Solution:",val
        print *,x0(1),x0(2)
        end subroutine

c this is the objective function
        subroutine func(n,x,f)
        double precision x(2)
        double precision f
        integer n
        f=x(1)*x(1)+x(2)*x(2)-cos(18*x(1))-cos(18*x(2))
        return
        end function

```

```
// wrapper.c
// just calls the Fortran program
extern int rastrigin();
int main() {
    return rastrigin();
}

# makefile =====
# compilers
CPP = g++
CC = gcc
FOR = g77

# Some options
CPPFLAGS = -g -Wno-deprecated
CFLAGS = -g
FORFLAGS =-g -fno-underscoring

# Object files for the example
OBJ3 = example2f.o wrapper.o

LIB_PATH = $(HOME)/ganso/
#LIB_PATH = /usr/local/lib/
INCLUDE_PATH = $(HOME)/ganso/
#INCLUDE_PATH = /usr/local/include/

all:    example2f

# we need g2c library and math library
example2f:$(OBJ3)
    $(CPP) -o example2f $(OBJ3) $(CPPFLAGS) \
    $(LIB_PATH)libganso.a -lm -lg2c

example2f.o:    example2f.f
    $(FOR) -c example2f.f $(FORFLAGS) -I$(INCLUDE_PATH)

wrapper.o: wrapper.c
    $(CC) -c wrapper.c $(CFLAGS) -I$(INCLUDE_PATH)
```

4.6.2 lf95 compiler

Lahey fortran can link with `gansodll.dll` compiled with Visual C++ by adding the command:

```
DLL_IMPORT name_of_the_function
```

to the code. When using a dll, the Lahey Fortran compiler does not append any underscore at the end of functions, and unlike in the rest of the Fortran code, it is case-sensitive. At compile time, the flag `-ml msvc` should be used to precise that the dll has been compiled with Visual C++.

Because the dll is directly used, lf95 does not require a C wrapper, and hence the user can have the standard fortran main program (not a subroutine). The example below can be compiled using the following command:

```
lf95 example2c.f -winconsole gansodll.lib -ml msvc
```

In the Lahey ED4W editor, the compiler options can be set up under: 'Tool—Programs', then selecting the 'General' button and editing the options for the appropriate compiler. Just add `gansodll.lib -ml msvc` to the command line.

```

      program Rastrigin
c     Import the DLL
      DLL_IMPORT minimizeecam
      DOUBLE PRECISION boundsL(2),boundsU(2),x0(2),val
      DOUBLE PRECISION A(3),B(1),rLipConst
      INTEGER retcode,nvar,basic(2),minimizeecam
c     To pass a subroutine or a function as a function pointer in C/C++
      EXTERNAL func
c     Define the box constraints
      nvar=2
      do i=1,nvar
      boundsL(i)=-1.
      boundsU(i)=1.
      x0(i)=0.5
      A(i)=1.
      end do
      B(1)=1.
      basic(1)=1
      basic(2)=2
c     Estimate of the Lipschitz constant
      rLipConst=40.
c     Calling the ECAM method
```

```
retcode=minimizeecam(2,x0,val,func,0,0,rLipConst,NULL
2,NULL,NULL,NULL,boundsL,boundsU,basic,3000,10)
print *,"ECAM Solution:",val
print *,x0(1),x0(2)
print *,val
end program

subroutine func(n,x,f)
double precision x(2)
double precision f
integer n
f=x(1)*x(1)+x(2)*x(2)-cos(18*x(1))-cos(18*x(2))
return
end subroutine
```


Chapter 5

Examples of usage

5.1 Sample code

There are several examples of the usage of `GANSO` provided in the distribution. The best way to use `GANSO` is to declare an instance of the class `Ganso` and call its members directly.

The user needs to define the objective function with three parameters as

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

Then include `ganso.h`, compile and link against `ganso` library by as described in Section 2.5, or as shown in the example makefile.

Note for Windows users: `gansodll.dll` is compiled using `__stdcall` option. The examples and user's calling programs should be compiled with this option as well. In Visual C++ compilers `__stdcall` **is not** the default option. The user needs to manually change the project settings (in `Project->Settings->C/C++->Code generation` from Microsoft-specific `__cdecl` to standard `__stdcall` calling convention. The same can be done by using flag `\Gz` in the make file.

Other compilers on Windows platform, such as `gcc` use a different binary format for dynamically linked libraries. There are a number of conversion techniques that can be found on the internet. For users who use `gcc` under `MinGW` system, static library `libganso.a` is provided, the user should follow the instructions related to unix/linux installation and linking.

Sample optimization problems

Problem 1

A problem in the chemical equilibrium at constant temperature and pressure from the book J. Bracken and G.P. McCormick, "Selected Applications of Nonlinear Programming." John Wiley & Sons, New York, 1968.

$$\begin{aligned} \text{minimize } f(x) &= \sum_{i=1}^{10} x_i \left(c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right) \\ \text{s.t. } x_1 + 2x_2 + 2x_3 + x_6 + x_{10} &= 2 \\ x_4 + 2x_5 + x_6 + x_7 &= 1 \\ x_3 + x_7 + x_8 + 2x_9 + x_{10} &= 1 \\ x_i &\geq 0, i = 1, \dots, 10. \end{aligned}$$

The vector c is given as

$$c = (-6.089, -17.164, -34.054, -5.9514, -24.721, -14.986, -24.1, -10.708, -26.662, -22.179).$$

A modification of this problem includes inequality constraint

$$x_3 + x_7 \leq 0.2.$$

In the code below constraints $x_i \geq 0.0001$ are used for the sake of simplicity of the objective function evaluation.

Problem 2

Rastrigin function Nr 1.

$$\begin{aligned} \text{minimize } f(x) &= x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2) \\ \text{s.t. } -1 &\leq x_i \leq 1, i = 1, \dots, 2. \end{aligned}$$

```

// example constrained optimization problem
#include "ganso.h"
// coefficients of the obj. function
double CC[10] = { -6.089, -17.164, -34.054, -5.9514, -24.721,
-14.986, -24.1, -10.708, -26.662, -22.179};
// implementation of the user's test objective function
void myfunction(int* n, double* x, double* f) {
    int i;
    double t1=0.0,objf=0.0;
    for(i=0;i<*n;i++) t1 +=x[i];
    for(i=0;i<*n;i++) objf +=x[i]*(CC[i]+log(fabs(x[i]/t1)) );
    *f=objf;
}

int main() {
// declare an instance of Ganso class, exported from the dll
    Ganso G;
// matrix of constraints (3 inequality constraints)
    double A[30] = {1,2,2,0,0, 1,0,0,0,1,
                    0,0,0,1,2, 1,1,0,0,0,
                    0,0,1,0,0, 0,1,1,2,1};
// vector of right-hand sides
    double B[3] = {2,1,1};

// optional equality constraint
    double C[10] = {0,0,1,0,0, 0,1,0,0,0};
// RHS
    double D[1] = {0.2};
// optional index of basic variables 10 - 3 = 7 basic variables
    int index[10]= {1,4,5,6,7,8,9,0,0,0};

    double boundsL[10],boundsU[10], x0[10],val;
    int i,j,retcode;
    for(i=0;i<10;i++) {
        x0[i]=-1.0; // infeasible starting point
        boundsL[i]=0.0001; // lower bounds
    }
// perform local minimization using DFBM, starting from x0.
    retcode=G.MinimizeDFBM(10,x0,&val,myfunction,3,0,A,NULL,B,NULL,
    boundsL, NULL, NULL,0);
}

```

```
// the same, but using an additional equality constraint  $x[2]+x[6]=0.2$ 
retcode=G.MinimizeDFBM(10,x0,&val,myfunction,3,1,A,C,B,D,
boundsL, NULL, NULL,0);

// now using combined DFBM+ECAM method, with auxiliary global search
// in 3 variables and 500 iterations
retcode=G.MinimizeDFBMECAM(10,x0,&val,myfunction,3,1,A,C,B,D,
boundsL, NULL, NULL,1000,500,3);

cout<<"The value at the minimum: "<<val<<endl;
cout<<"The minimizer is: "<<endl;
for(i=0;i<10;i++) cout<<x0[i]<<" ";
cout<< endl;
return 0;
}
```

```

// example of box constrained global optimization problem
#include <cstdio>
#include <cmath>
#include <iostream>
using namespace std;

#include "ganso.h"

void myfunction(int* n, double* x, double* f) {
    *f=x[0]*x[0] + x[1]*x[1] - cos(18 *x[0])- cos(18 *x[1]);
}

int main() {
// declare an instance of Ganso class
    Ganso G;
// define the box constraints
    double boundsL[2] = {-1,-1};
    double boundsU[2] = {1,1};
    double x0[2],val, LipConst=40;
    // Estimate of the Lipschitz constant
    int retcode;
// ECAM global search with the limit of 3000 iterations.
    retcode=G.MinimizeECAM(2,x0,&val,myfunction,0,0,40,
        NULL,NULL,NULL,NULL, boundsL, boundsU, NULL,3000,10);
    cout<<"ECAM solution: "<<val<<endl<<x0[0]<<" "<<x0[1]<<endl;

// same but using a combination ECAM+DFBM, only 100 iterations
// short form
    retcode=G.MinimizeECAMDFBM_0(2,x0,&val,myfunction,40, boundsL, boundsU,100);
    cout<<"ECAM+DFBM solution: "<<val<<endl<<x0[0]<<" "<<x0[1]<<endl;

// Simple random start method, use 100 random initial points
    retcode=G.MinimizeRandomStart_0(2,x0,&val,myfunction,
        boundsL, boundsU,100);
    cout<<"Random Start solution: "<<val<<endl<<x0[0]<<" "<<x0[1]<<endl;

    return 0;
}

```

5.2 Linking

To link against `GANSO` library use the example makefile provided with your distribution. If you use Windows version, ensure that the `gansodll.dll` file is in the same directory as your main program, or on the path. See the `readme` file coming with your distribution.

Note for Windows users: `gansodll.dll` is compiled using `__stdcall` option. The examples and user's calling programs should be compiled with this option as well. In Visual C++ compilers `__stdcall` is **not** the default option. The user needs to manually change the project settings (in `Project->Settings->C/C++->Code generation` to from Microsoft-specific `__cdecl` to standard `__stdcall` calling convention. The same can be done by using flag `\Gz` in the make file.

Users of `gcc` under `MinGW` should use static library and follow unix/linux instructions. The makefile in the `examples` directory can be the starting point.

5.3 Tips

- Always scale your problem, including the objective function and constraints. Try to scale objective function to a reasonable range, say $[-100,100]$. Scale variables to a range around $[-10,10]$. This will help the algorithms deliver better accuracy and stability.
- Ensure the constraints are linearly independent. The algorithm will return an error otherwise.
- Use adequate parameters. All algorithms are generic, they can be executed with any dimension or number of iterations, but be realistic in your expectations. ECAM can be safely used for up to 6-8 variables, but its complexity grows very quickly with the number of iterations and dimension. Use about 10000 iterations, if more, be prepared for a long running time and large memory demand. If using ECAM+DFBM or RandomStart, at each iteration expensive local search by DFBM will be performed – be prepared to wait. When using DFBM+ECAM, use small dimensional subspace for ECAM (< 10).
- Do not unnecessarily specify box constraints for DFBM, especially the

upper bound. It works faster with fewer constraints. However all other methods do require box constraints.

- Always try the methods on simpler problems before using them for production runs. Estimate the running time/memory requirements using a small number of iterations.
- Specifying negative *maxiter* parameter in all methods (*speed* parameter in `MinimizeDSO` and `MinimizeIterativeDSO`) has the effect of not using DFBM to improve the solution at the last stage. This will return the minimum found by the global algorithm (ECAM or DSO), not improved by local search.

5.4 Where to get help

The software library **GANSO** and its components, are distributed by CIAO AS IS, with no warranty, explicit or implied, of merchantability or fitness for a particular purpose. CIAO will provide limited technical support for registered users, by electronic media. CIAO, at its sole discretion, may provide advice to registered users on the proper use of **GANSO** and its components.

Any queries regarding technical information, sales and licensing should be directed to j.ugon@ballarat.edu.au. For updates check <http://www.ganso.com.au>

Bibliography

- [Bag02] A. Bagirov. A method for minimization of quasidifferentiable functions. *Optimization Methods and Software*, 17:31–60, 2002.
- [Bag03] A. Bagirov. Continuous subdifferential approximations and their applications. *Journal of Mathematical Sciences*, 115:2567–2609, 2003.
- [BB02] L.M. Batten and G. Beliakov. Fast algorithm for the cutting angle method of global optimization. *Journal of Global Optimization*, 24:149–161, 2002.
- [Bel04] G. Beliakov. The cutting angle method - a tool for constrained global optimization. *Optimization Methods and Software*, 19:137–151, 2004.
- [Bel05] G. Beliakov. A review of applications of the cutting angle methods. In A. Rubinov and V. Jeyakumar, editors, *Continuous Optimization: Current Trends and Modern Applications*, pages 209–248. Springer, New York, 2005.
- [BGLS00] J.F. Bonnans, J.C. Gilbert, C. Lemarechal, and C.A. Sagastizabal. *Numerical Optimization. Theoretical and Practical Aspects*. Springer, Berlin, Heidelberg, 2000.
- [BR01] A. Bagirov and A. Rubinov. Modified versions of the cutting angle method. In N. Hadjisavvas and P.M. Pardalos, editors, *Convex analysis and global optimization*, volume 54 of *Nonconvex optimization and its applications*, pages 245–268. Kluwer, Dordrecht, 2001.

- [Cla83] F.H. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley, New York, 1983.
- [DR86] V. F. Demyanov and A. Rubinov. *Quasi-differential Calculus*. Optimization Software, Inc., New York, 1986.
- [DR95] V.F Demyanov and A.M. Rubinov. *Constructive Nonsmooth Analysis*. Peter Lang, Frankfurt am Main, 1995.
- [HP95] R. Horst and P. M. Pardalos. *Handbook of Global Optimization*. Nonconvex Optimization and its Applications. Kluwer Academic Publishers, Dordrecht; Boston, 1995.
- [HPT00] R. Horst, P. Pardalos, and N.V. Thoai, editors. *Introduction to Global Optimization*, volume 48 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, 2nd edition, 2000.
- [HT93] R. Horst and H. Tuy. *Global optimization: deterministic approaches*. Springer-Verlag, Berlin; New York, 2nd rev. edition, 1993.
- [Mam04] M.A. Mammadov. A new global optimization algorithm based on dynamical systems approach. In A. Rubinov and M. Sniedovich, editors, *6th Intl Conf. on Optimization: Techniques and Applications*, Ballarat, 2004. University of Ballarat.
- [MO05] M.A. Mammadov and R. Orsi. H.infinity systhesis via a nonsmooth, nonconvex optimization approach. *Pacific Journal of Optimization*, 1:405–420, 2005.
- [MRY05] M.A. Mammadov, A. Rubinov, and J. Yearwood. Dynamical systems described by relational elasticities with applications to global optimization. In A. Rubinov and V. Jeyakumar, editors, *Continuous Optimisation: Current Trends and Modern Applications*, pages 365–385. Springer, New York, 2005.
- [MW93] J. Moré and S. J. Wright. *Optimization Software Guide*. SIAM, Philadelphia, 1993.

- [Pin96] J. Pintér. *Global Optimization in Action: Continuous and Lipschitz Optimization—Algorithms, Implementations, and Applications*. Nonconvex optimization and its applications; v. 6. Kluwer Academic Publishers, Dordrecht; Boston, 1996.
- [Roc70] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, 1970.
- [Rub00] A.M. Rubinov. *Abstract Convexity and Global Optimization*, volume 44 of *Nonconvex optimization and its applications*. Kluwer Academic Publishers, Dordrecht; Boston, 2000.
- [SS00] R. G. Strongin and Y. D. Sergeyev. *Global Optimization with Nonconvex Constraints: Sequential and Parallel Algorithms*. Nonconvex optimization and its applications; v.45. Kluwer Academic, Dordrecht; London, 2000.