TECHNISCHE UNIVERSITÄT
TUHH
HAMBURG-HARBURG

Berichte des
Forschungsschwerpunktes
Informations- und
Kommunikationstechnik

Olaf Knüppel

A PROFIL/BIAS Implementation of a Global Minimization Algorithm

Oktober 1995 Bericht 95.4

Olaf Knüppel: Oktober 1995

A PROFIL/BIAS Implementation of a Global Minimization Algorithm

Bericht 95.4

IMPORTANT NOTE:

Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

Prof. Dr. S. M. Rump
Technische Universität Hamburg-Harburg
Technische Informatik III
D-21071 Hamburg
Germany

This report is available
via anonymous ftp from:
`ti3sun.ti3.tu-harburg.de`
in the file:
`/pub/reports/report95.4.ps.Z`

A PROFIL/BIAS Implementation of a Global Minimization Algorithm

Olaf Knüppel

Technische Informatik III
Technische Universität Hamburg-Harburg
D–21071 Hamburg

email: knueppel@tu-harburg.d400.de

October 6, 1995

Abstract

After several publications about our new global optimization method, we decided to provide the public with a simple implementation of the new method. This implementation has been written using PROFIL/BIAS together with the PROFIL extensions. Some example programs which show some simple applications of the method are also contained in the package. On the one hand, the sample programs give a good example of how to use automatic differentiation. On the other hand, a little bit more sophisticated solutions where speed is of most interest are also presented. In the latter case, the user is free to write special subroutines for computing the derivatives.

This is a brief description of the implementation of the global minimization method and its usage. It is neither intended to explain the specific implementation details of the package nor to explain the mathematical background.

Contents

1 Introduction **3**

2 Installation **3**

2.1 Installing the Library and Include Files 4

3 The Method **4**

3.1 Description of Modules 4

3.2 Simple Driver Interface 5

3.3 The More Sophisticated Driver 6

4 Sample Programs **9**

4.1 Example 1 — SimpleTestOpt.C 9

4.2 Example 2 — TestOpt.C 11

Bibliography **14**

1 Introduction

This documentation describes in a compact way the usage of the global minimization algorithm whose mathematical background together with many applications is described in detail in [7, 6, 3, 2, 5, 1, 4, 8]. The implementation of the algorithm is based on PROFIL/BIAS [9, 10] together with the PROFIL/BIAS Extensions [11].

In some cases it is assumed that the reader is familiar with basic C programming techniques and terms as header files, modules, libraries, and so forth.

Unexperienced readers may take the sample programs and test and modify them.

2 Installation

The installation of the global minimization package is similar to the installation of the PROFIL extensions. As a prerequisite it is assumed that PROFIL, BIAS, and the PROFIL extensions are properly installed [10, 11]. To install the global minimization package, change to the PROFIL directory and copy the package file into this directory. The package file is called `profopt.tar.Z` under Unix systems and `profopt.tgz` under DOS or OS/2. To unpack the compressed package archive, type

```
zcat profopt.tar.Z | tar xf -
```

under Unix or

```
gzip -dc profopt.tgz | tar xf -
```

under DOS or OS/2. After that the compressed package (`profopt.tar.Z` or `profopt.tgz`) may be deleted.

Type `cd GlobOpt` to change to the directory containing the global minimization subroutines. Depending on your compiler, type the following to build all libraries:

Command	Architecture	Compiler used
`dmake xlC`	IBM RS/6000	IBM xlc
`dmake ibmgcc`	IBM RS/6000	gcc
`dmake hppaCC`	HP 9000/700	HP CC
`dmake hppagcc`	HP 9000/700	gcc
`dmake sun4CC`	Sparc	SUN CC
`dmake sun4gcc`	Sparc	gcc
`dmake pcgcc`	80x86	gcc

After the `dmake` finishes, the library `ProfilOpt` has been built.

2.1 Installing the Library and Include Files

Until now, the library and the include files of the global minimization package are contained in the `GlobOpt` directory. To install them in a different place, please refer to [10], where the appropriate steps are described for the PROFIL distribution.

Due to limitations in file name length on some systems, some of the file names mentioned below may be different on your system. If any differences occur, those can be identified in the file `names` where the first name in each line corresponds to the name mentioned in this documentation and the second name is the file name on your system. On such systems for example the library `ProfilOpt` is called `propt` instead (i.e. the file name is `libpropt.a`).

3 The Method

In order to keep this description small, we assume that the reader is familiar with the basics of interval mathematics as well as with the principles behind the new method (see, e.g. [7, 6, 3, 2, 5, 1, 4, 8]). The implementation combines the derivative-free method together with the version using derivatives. It is up to the user which version is used.

The global optimization method is available in two versions, where the first is an easy-to-use simple driver for the second more sophicticated version. The first version uses automatic differentiation for the computation of the derivatives. Therefore the function only has to be written once. The disadvantage of this approach is that it is more slowly, because with each function evaluation a gradient is also computed. Additionally, all real valued function/derivative computation is done using an interval function or derivative call. On the other hand, it is easier to use this simple approach. The second version needs special subroutines for each function/derivative type. Therefore, 4 or 2 (if not using derivatives) different subroutines have to be written. The advantage of this approach lies in the flexibility when writing the subroutines. For example, the user is free to use hand-optimized inclusion functions.

3.1 Description of Modules

The PROFIL/BIAS implementation of the global minimization method is divided into several modules which also might be helpful for other purposes. The modules are:

`AppList`: Implementation of lists containing approximations of local and/or global minima amlong with additional information. This list is called "A" in the mathematical description of the method. The list data type is called `APPROXIMATIONLIST`.

`VecList`: Implementation of lists containing boxes to be processed later. This list is called "S" in the mathematical description of the method. The list data type is called `SOLUTIONLIST`.

`Expand`: The expansion strategy.

`SimpleOptDriver`: The simple driver for the method.

`UnconstrainedOpt`: The expert driver for the method. As local descent method, the method of Brent taken from the PROFIL/BIAS extension package is used. The call of the descent method is hardcoded into the file `UnconstrainedOpt.C`, but can be changed, if another method is desired.

`VecUtils`: Some necessary utility subroutines.

In the following, we describe the two versions of the interface to the global minimization method in detail, starting with the simple driver.

3.2 Simple Driver Interface

The file `SimpleOptDriver.h` contains all necessary definitions for the simple driver interface.

```
VOID SimpleOptDriver (INT Iterations, INT BranchLevels,
                      INTERVAL_AUTODIFF (*TheFunction)(INTERVAL_VECTOR &),
                      INTERVAL_VECTOR TestDomain,
                      SOLUTIONLIST & SolutionList,
                      APPROXIMATIONLIST & ApproximationList,
                      INTERVAL & ValueBounds,
                      REAL Tolerance,
                      REAL RaiseFactor,
                      REAL HeuristicExpandFactor);
```

`Iterations`: The maximal number of iterations. This parameter is called n_{it} in the mathematical description of the method.

`BranchLevels`: The maximal number of bisection performed in each coordinate direction in each iteration. This parameter is called n_{d} in the mathematical description of the method.

`TheFunction`: a pointer to the function to be minimized. This function must return and inclusion for the function value, gradient, and Hessian for a given range. All return values are encoded in the type `INTERVAL_AUTODIFF`. For an example of how to write such a function, see example section below.

`TestDomain`: The domain of the function.

`SolutionList`: Contains all unprocessed boxed on output. The list must be empty on input.

`ApproximationList`: Contains all computed approximations together with additional information (e.g. enclosures of the minima). The list must be empty on input.

`ValueBounds`: Contains bounds for the function value on output.

`Tolerance`: Relative tolerance for the local descent method. This parameter has a default value of 10^{-6}.

`RaiseFactor`: Parameter of the method. Larger values will increase the number of calls of the local descent method. This parameter has a default value of 0.2 which is also used for all examples published with the method.

`HeuristicExpandFactor`: Parameter of the method. Larger values will enlarge the heuristic area around computed local minima. Larger heuristic areas might decrease the number of calls of the local descent method. This parameter has a default value of 0.2 which is also used for all examples published with the method.

3.3 The More Sophisticated Driver

The more sophisticated driver allows specialized subroutines for each function or derivative required. Especially it is possible to use hand-optimized code for the inclusion functions. Even functions, which need special enclosure evaluations (e.g. eigenvalue problems) could be solved with this driver. Additionally, a more complex interactive handling of the method is possible, i.e. after some iterations, the solutions could we examined and depending on the result, more iteration steps could be performed. This makes this version more suitable for more difficult functions.

The sophisticated driver consists of 3 subroutines:

`StartUnconstrainedOptimization` This subroutine is called when initiating the minimization method. It sets up all internal data structures and calls the next subroutine.

`UnconstrainedOptimization` This subroutine assumes that all internal data structures have been set up and performs the demanded number of iteration steps. After it has finished, additional iterations can be achieved by calling this subroutine again (perhaps with modified parameters).

`CleanUpLists` As a finishing step, after all requested iterations have been performed, the lists can be cleaned from unnecessary elements by calling this subroutine.

In the following, we will describe the parameters of those 3 subroutines in detail:

```
VOID StartUnconstrainedOptimization(
  SOLUTIONLIST & SolutionList,
  APPROXIMATIONLIST & ApproximationList,
  INT Iterations, INT BranchLevels, REAL Eps,
  REAL RaiseFactor, REAL HeuristicExpandFactor,
  REAL & LowerBound, REAL & UpperBound,
  INTERVAL_VECTOR & TheDomain,
  REAL (* TheRealFunction)(VECTOR &),
  INTERVAL (* TheIntervalFunction)(INTERVAL_VECTOR &),
  INTERVAL_VECTOR (* TheIntervalGradient)(INTERVAL_VECTOR &),
  INTERVAL_MATRIX (* TheIntervalHessian)(INTERVAL_VECTOR &));
```

`SolutionList:` Contains all unprocessed boxed on output. The list must be empty on input.

`ApproximationList:` Contains all computed approximations together with additional information (e.g. enclosures of the minima). The list must be empty on input.

`Iterations:` The maximal number of iterations. This parameter is called n_{it} in the mathematical description of the method.

`BranchLevels:` The maximal number of bisection performed in each coordinate direction in each iteration. This parameter is called n_{d} in the mathematical description of the method.

`Eps:` Relative tolerance for the local descent method.

`RaiseFactor:` Parameter of the method. Larger values will increase the number of calls of the local descent method.

`HeuristicExpandFactor`: Parameter of the method. Larger values will enlarge the heuristic area around computed local minima. Larger heuristic areas might decrease the number of calls of the local descent method.

`LowerBound`: Contains the lower bound for the function value on output.

`UpperBound`: Contains the upper bound for the function value on output.

`TheDomain`: The domain of the function.

`TheRealFunction`: a pointer to the real-valued function to be minimized.

`TheIntervalFunction`: a pointer to the interval-valued function to be minimized.

`TheIntervalGradient`: a pointer to an inclusion function of the gradient. If no derivatives are available, the parameter should be left empty. In this case, the derivative-free version of the method is used.

`TheIntervalHessian`: a pointer to an inclusion function of the Hessian (if derivatives are available).

To perform further steps after the initial iterations, the following subroutine is used:

```
VOID UnconstrainedOptimization (
  SOLUTIONLIST & SolutionList,
  APPROXIMATIONLIST & ApproximationList,
  INT Iterations, INT BranchLevels, REAL Eps,
  REAL RaiseFactor, REAL HeuristicExpandFactor,
  REAL & LowerBound, REAL & UpperBound,
  BOOL UseDerivatives);
```

Any parameters not described here have the same meaning as in `StartUnconstained-Optimization`.

`SolutionList`: Contains all unprocessed boxed on input and returns the new list on output.

`ApproximationList`: Contains all computed approximations together with additional information (e.g. enclosures of the minima) on input and returns the new list on output.

`LowerBound`: Contains a lower bound for the function value on input and yields the new computed lower bound on output.

`UpperBound`: Contains an upper bound for the function value on input and yields the new computed lower bound on output.

`UseDerivatives`: Must be set to `TRUE`, if derivatives should be used. This parameter has a default value of `FALSE`.

To clean up the lists, the following subroutine could be used:

```
VOID CleanUpLists (
  SOLUTIONLIST & SolutionList,
  APPROXIMATIONLIST & ApproximationList,
  REAL Eps, REAL LowerBound, REAL UpperBound);
```

`SolutionList`: Contains all unprocessed boxed on input and returns the new list on output.

`ApproximationList`: Contains all computed approximations together with additional information (e.g. enclosures of the minima) on input and returns the new list on output.

`Eps`: Relative tolerance for the local descent method.

`LowerBound`: Contains a lower bound for the function value on input.

`UpperBound`: Contains an upper bound for the function value on input.

4 Sample Programs

The subdirectory `Examples` contains sample programs for some of the new features discussed in this documentation. The examples are kept quite simple to make the main ideas of the new features clear.

All sample programs can simply be build by appending `_example` to the `dmake` command discussed in the installation section. For example,

```
dmake xlC_examples
```

builds the library (if not already done) and all sample programs.

4.1 Example 1 — SimpleTestOpt.C

```
/************************************************************************
*  SimpleTestOpt.C (Simple Test Program for Minimization Algorithm)
*
*  Version:                    1.4
```

```
*  Date:                        11/30/94
*  Author:                      O. Knueppel
**************************************************************************/

//
// This test programs uses automatic differentiation to
// compute the derivatives of the function.
// It is a simple example for the easy-to-use driver "SimpleOptDriver".
//

#include "SimpleOptDriver.h"
#include "Functions.h"

//
// The Branin function is used as test function.
// This function has 5 global minima.
//

INTERVAL_AUTODIFF TestFunction (INTERVAL_VECTOR & v)
{
  INTERVAL_AUTODIFF x(v); /* Define independent variables */

  return Sqr (1.0 - 2.0 * x(2) + 0.05 * Sin (4.0 * Constant::Pi * x(2)) - x(1))
       + Sqr (x(2) - 0.5 * Sin (2.0 * Constant::Pi*x(1)));
}

VOID SetTestDomain (INTERVAL_VECTOR & x)
// The domain of the test function
{
  Resize (x, 2);
  x(1) = Hull (-10.0, 10.0);
  x(2) = Hull (-10.0, 10.0);
}


INT main()
{
  INT Iterations, BranchLevels;
  SOLUTIONLIST SolutionList;
  APPROXIMATIONLIST ApproximationList;
  INTERVAL_VECTOR TestDomain;
```

```
  INTERVAL ValueBounds;

  SetTestDomain (TestDomain);

  cout << "Computing all global minima of the Branin function" << endl << endl;

  cout << "To get all minima, use e.g. nit = 6, nd = 3" << endl << endl;

  cout << "nit = "; cin >> Iterations;
  cout << "nd  = "; cin >> BranchLevels;

  SimpleOptDriver (Iterations, BranchLevels, TestFunction, TestDomain,
   SolutionList, ApproximationList, ValueBounds);

  cout << "f min in " << ValueBounds << endl;
  cout << "Solution List:" << endl << SolutionList << endl;
  cout << "Approximation List:" << endl << ApproximationList << endl;
  return 0;
}
```

4.2 Example 2 — TestOpt.C

```
/*************************************************************************
*  TestOpt.C (Simple Test Program for Minimization Algorithm)
*
*  Version:                    1.6
*  Date:                       11/30/94
*  Author:                     O. Knueppel
*************************************************************************/

#include "UnconstrainedOpt.h"
#include "Functions.h"

//
// This simple test program displays how to use different subroutines
// for the computation of the function/derivative values. For a
// more simple, but also slower approach see "SimpleTestOpt.C".
//

//
// The Rastrigin function [Rastrigin 1974] is used as test function.
```

```
// This function has one global minimum:
//
//  x = (0,0), y = -2
//

VOID SetTestDomain (INTERVAL_VECTOR & x)
// The domain of the test function
{
  Resize (x, 2);
  x(1) = Hull (-1.0, 1.0);
  x(2) = Hull (-1.0, 1.0);
}

REAL RealTestFunction (VECTOR & x)
// real valued test function
{
  return Sqr (x(1)) + Sqr (x(2)) - Cos (18.0 * x(1)) - Cos (18.0 * x(2));
}

INTERVAL IntervalTestFunction (INTERVAL_VECTOR & x)
// interval valued test function
{
  return Sqr (x(1)) + Sqr (x(2)) - Cos (18.0 * x(1)) - Cos (18.0 * x(2));
}

INTERVAL_VECTOR IntervalTestGradient (INTERVAL_VECTOR & x)
// interval valued gradient of test function
{
  INTERVAL_VECTOR t(2);

  t(1) = 2.0 * x(1) + 18.0 * Sin (18.0 * x(1));
  t(2) = 2.0 * x(2) + 18.0 * Sin (18.0 * x(2));
  return t;
}

INTERVAL_MATRIX IntervalTestHessian (INTERVAL_VECTOR & x)
// interval valued Hessian of test function
{
  INTERVAL_MATRIX t(2,2);

  t(1,1) = 2.0 + 324.0 * Cos (18.0 * x(1)); t(1,2) = 0.0;
```

```
  t(2,1) = 0.0; t(2,2) = 2.0 + 324.0 * Cos (18.0 * x(2));
  return t;
}

INT main()
{
  INT Iterations, BranchLevels;
  SOLUTIONLIST SolutionList;
  APPROXIMATIONLIST ApproximationList;
  INTERVAL_VECTOR TestDomain;
  REAL LowerBound, UpperBound;

  SetTestDomain (TestDomain);

  cout << "Computing all global minima of the Rastrigin function" << endl << endl;

  cout << "nit = "; cin >> Iterations;
  cout << "nd  = "; cin >> BranchLevels;

  StartUnconstrainedOptimization (SolutionList, ApproximationList,
  Iterations, BranchLevels, 1E-6, 0.2, 0.2,
  LowerBound, UpperBound, TestDomain,
  RealTestFunction, IntervalTestFunction,
  IntervalTestGradient, IntervalTestHessian);

  CleanUpLists (SolutionList, ApproximationList, 1e-6, LowerBound, UpperBound);

  cout << "f min in " << Hull (LowerBound, UpperBound) << endl;
  cout << "Solution List:" << endl << SolutionList << endl;
  cout << "Approximation List:" << endl << ApproximationList << endl;
  return 0;
}
```

References

[1] C. Jansson. A global minimization method: The one-dimensional case. Bericht 91.2, TU Hamburg-Harburg, Technische Informatik III, Nov. 1991.

[2] C. Jansson. A global optimization method using interval arithmetic. In *Computer Arithmetic and Enclosure Methods*, pages 259–268. Elsevier Science Publishers B.V., North-Holland, 1992.

[3] C. Jansson. On self-validating methods for optimization problems. In J. Herzberger, editor, *Topics in Validated Computations*, Studies in Computational Mathematics, Amsterdam, 1994. North Holland, Zur Veröffentlichung eingereicht.

[4] C. Jansson, O. Knüppel. A global minimization method: The multi-dimensional case. Bericht 92.1, TU Hamburg-Harburg, Technische Informatik III, Jan. 1992.

[5] C. Jansson, O. Knüppel. A minimization method for systems with singular value constraints. In *Operations Research '92*, pages 190–192. Physica-Verlag, Heidelberg, 1992.

[6] C. Jansson, O. Knüppel. A branch and bound algorithm for bound constrained optimization problems. *Zur Veröffentlichung eingereicht*, 1993.

[7] C. Jansson, O. Knüppel. Eine intervallanalytische Methode für globale Optimierungsprobleme. *Zeitschrift für angewandte Mathematik und Mechanik*, 73(6):T741–T743, 1993.

[8] C. Jansson, O. Knüppel. Numerical results for a self-validating global optimization method. Bericht 94.1, TU Hamburg-Harburg, Technische Informatik III, Feb. 1994.

[9] O. Knüppel. BIAS — Basic Interval Arithmetic Subroutines. Bericht 93.3 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TU Hamburg-Harburg, TU Hamburg-Harburg, Technische Informatik III, July 1993.

[10] O. Knüppel. PROFIL — Programmer's Runtime Optimized Fast Interval Library. Bericht 93.4 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TU Hamburg-Harburg, TU Hamburg-Harburg, Technische Informatik III, July 1993.

[11] O. Knüppel, T. Simenec. PROFIL/BIAS extensions. Bericht 93.5 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TU Hamburg-Harburg, TU Hamburg-Harburg, Technische Informatik III, Nov. 1993.

In dieser Reihe sind bisher erschienen:

Bericht 90.1 (Juni 1990)
S. M. Rump: Approximate inverses of almost singular matrices still contain useful information
Bericht 90.2 (September 1990)
C. Jansson: A Posteriori Error Bounds for the Solution of Linear Systems with Simplicial Techniques
Bericht 90.3 (September 1990)
S. M. Rump: A class of arbitrarily ill-conditioned floating-point matrices
Bericht 90.4 (Oktober 1990)
C. Jansson: A Fast Direct Method for Computing Verified Inclusions
Bericht 90.5 (November 1990)
S. M. Rump: On the Solution of Interval Linear Systems
Bericht 91.1 (April 1991)
D. Husung: TPX Precompiler für Turbo Pascal
Bericht 91.2 (November 1991)
C. Jansson: A Global Minimization Method: The One-Dimensional Case
Bericht 91.3 (Dezember 1991)
S. M. Rump: Inclusion of the Solution for Large Linear Systems with M-Matrix
Bericht 92.1 (Januar 1992)
C. Jansson, O. Knüppel: A Global Minimization Method: The Multi-Dimensional Case
Bericht 92.2 (Februar 1992)
H.-P. Jüllig: Algorithmen mit Ergebnisverifikationen mit C++/2.0
Bericht 92.3 (Februar 1992)
S. M. Rump: Recent Results in Interval Mathematics
Bericht 92.4 (Februar 1992)
D. Husung: TPX Version 1.1 US Precompiler for Turbo Pascal 5.0
Bericht 92.5 (März 1992)
D. M. Claudio, S. M. Rump: Inclusion methods for real and complex functions in one variable
Bericht 92.6 (März 1992)
H.-P. Jüllig: BIBINS/2.0 C++-Bibliotheken für Vektoren und Matrizen über beliebigem skalaren Datentyp unter Berücksichtigung spezieller spärlicher Strukturen sowie Intervalldatentypen
Bericht 93.1 (Januar 1993)
S. M. Rump: Validated Solution of Large Linear Systems
Bericht 93.2 (Januar 1993)
D. Husung: TPX Version 1.1 US Precompiler for Turbo Pascal 5.0
Bericht 93.3 (Juli 1993)
O. Knüppel: BIAS — Basic Interval Arithmetic Subroutines
Bericht 93.4 (Juli 1993)
O. Knüppel: PROFIL — Programmer's Runtime Optimized Fast Interval Library
Bericht 93.5 (November 1993)
O. Knüppel, T. Simenec: PROFIL/BIAS Extensions
Bericht 93.6 (November 1993)
O. Knüppel: A Multiple Precision Arithmetic for PROFIL
Bericht 93.7 (Dezember 1993)
S. M. Rump: Verification Methods for Dense and Sparse Systems of Equations
Bericht 94.1 (Februar 1994)
C. Jansson, O. Knüppel: Numerical Results for a Self-Validating Global Optimization Method
Bericht 94.2 (Juni 1994)
C. Jansson: On Self-Validating Methods for Optimization Problems
Bericht 94.3 (Dezember 1994)
C. Jansson: Some Properties of Linear Interval Systems and Applications
Bericht 95.1 (März 1995)
S. M. Rump: Improved Iteration Schemes for Validation Algorithms for Dense and Sparse Nonlinear Systems
Bericht 95.2 (März 1995)
S. M. Rump: Expansion and Estimation of the Range of Nonlinear Functions
Bericht 95.3 (Mai 1995)
S. M. Rump: Bounds for the Componentwise Distance to the Nearest Singular Matrix