Optimization Methods and Software
Vol. 00, No. 00, January 2008, 1–10

Global Optimization Software

INTLAB implementation of an interval global optimization algorithm

László Pál[a] and Tibor Csendes[b]*

[a]*Faculty of Business and Humanities, Sapientia University, Miercurea-Ciuc, Romania*;
[b]*Institute of Informatics, University of Szeged, Szeged, Hungary*

(*May 2008*)

We describe a new implementation of an interval optimization algorithm with focus on the software related issues. The algorithm implemented in MATLAB that uses the INTLAB package supporting interval calculations and automatic differentiation solves the bound constrained global optimization problem. The method itself is a simplified version of those interval techniques much investigated in the past, which were first developed from the global optimization algorithm of the Numerical Toolbox for Verified Computing. According to the numerical studies completed, the new, INTLAB based implementation is closely as efficient as its C-XSC-based basis algorithm – with the exception of the CPU time needed (the longer computations are due to the interpreter nature of MATLAB).

Keywords: interval methods; INTLAB; bound constrained global optimization; verified solution

AMS Subject Classification: 65K05, 90C30

1. Introduction

Bound constrained global optimization problems in the form of

$$\min_{x \in X} f(x)$$

are common with $X = \{x_i \in [\underline{x}_i, \overline{x}_i],\ i = 1, \ldots, n\}$, and $\underline{x}_i, \overline{x}_i \in \mathbb{R},\ i = 1, \ldots, n$. In several cases we can assume that the objective function, f is smooth. Our algorithm utilizes smoothness, but with skipping the interval Newton step and the concavity test, it can also be applied for nonsmooth objective functions.

Just to name some of the numerous applications of global optimization, we point on some of our recent publications: we solved with such techniques hard mathematical problems arising in the field of qualitative analysis of dynamical systems [2, 5, 6] and discrete geometry, for optimal packing of circles in the square [10, 13]. Global optimization methods have also been applied for theoretical chemical problems [1], and for the evaluation of bounding methods [14].

MATLAB is a natural environment for algorithm development and testing. Our aim was to provide an easy to use reliable global optimization method. We have recently completed a similar successful implementation in MATLAB for the stochastic GLOBAL procedure [7].

*Corresponding author. Email: Csendes@inf.u-szeged.hu

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/1055678xxxxxxxxxxxxx
http://www.informaworld.com

The algorithm investigated now uses only a subroutine calculating the objective function as information on the global optimization problem, i.e. the expression is not required. The procedure applies the gradient and the Hessian of the objective function, these are computed by the automatic differentiation facility of INTLAB.

2. The algorithm and its implementation

The branch-and-bound type method we have implemented is described by Algorithm **1**. This technique originates in the Numerical Toolbox for Verified Computing [8], and it applies the most common accelerating devices: the cutoff test, the concavity test, the monotonicity test, and the interval Newton step. Beyond natural interval extension (based on naive interval arithmetic), a simple centered form inclusion function is also applied. Once the inclusion of the gradient is available, the intersection of these inclusion functions proved to be a good quality estimation of the range of the objective function.

We use also multisection and advanced subdivision direction selection [9], albeit without those based on the pf^* heuristic algorithm parameter [3]. These later techniques will be inserted in the future: the present version is planed to be simple and easy to use. Multisection means this time that each interval will be subdivided into three subintervals according to the most promising two coordinate directions. The subdivision directions are determined according to the well tested and effective C subdivision direction selection rule (also used in [3] and [9]). The algorithm solves also one-dimensional problems. Then the described multisection technique is substituted by greedy bisection made again on the basis of the C rule.

For the MATLAB/INTLAB implementation we have followed closely the C-XSC code which was developed for bound constrained global optimization by Mihály Csaba Markót based on the algorithm documented in [11]. The control structures of the two algorithms are identical, while the vectorial array statements of MATLAB were applied wherever possible.

To use the new method, first install the INTLAB package for interval arithmetic based inclusion functions and verified numerical techniques. Download the compressed archive from `http://www.ti3.tu-harburg.de/rump/intlab/`, and follow the included guide and instructions. The installation requires a few minutes and some level of experience in operating system script programming. Otherwise the hints given in the user guide are sufficient. INTLAB is free for private use and for purely academic purposes provided proper reference is given acknowledging that the software package INTLAB has been developed by Siegfried M. Rump at Hamburg University of Technology, Germany [12]. INTLAB applies a sophisticated rounding that depends closely on the actual hardware. This is why it is easy to implement on a standard PC (also in Linux), while it cannot be used immediately on some modern workstations.

When the INTLAB package has been downloaded, decompress the archive, and place the files into a directory, that should be then given as the default directory, where MATLAB finds the INTLAB related files. This can be accomplished by setting the Current Directory properly at the top center position of the MATLAB window. The next step is to run the script `startintlab.m` from the main directory:

```
>> startintlab
```

which initiates various global variables and will do much of the rest. In case everything went well, no error message is obtained. Otherwise the user obtains the most important hints how to complete the implementation procedure. Note that

Algorithm 1 The bound constrained global optimization algorithm investigated

GlobalOptimize (f, X, ε, L_{res}, f^*)
$Y := X; \widetilde{f} := \overline{f}(m(X)); L_{res} := \{\}; L_{work} := \{\}; L_{temp} := \{\};$
repeat
  OptimalComponents(Y, k_1, k_2);
  Trisection$(Y, k_1, k_2, U^1, U^2, U^3)$;
  for $i := 1$ **to** 3 **do**
    if MonotonicityTest$(\nabla f(U^i))$ **then next** i;
    $f_U := f(U^i)$;
    if $\widetilde{f} < \underline{f_U}$ **then next** i;
    $f_U := f_U \cap$ CenteredForm$(U^i, \nabla f(U^i))$;
    if $\overline{f}(m(U^i)) < \widetilde{f}$ **then** $\widetilde{f} := \overline{f}(m(U^i))$;
      $L_{work} :=$ CutOffTest$(L_{work}, \widetilde{f})$;
    if $\widetilde{f} >= \underline{f_U}$ **then** $L_{temp} := L_{temp} \cup (U^i, \underline{f_U})$;
  if length$(L_{temp}) = 1$ **then**
    $U :=$ Head(L_{temp});
    if not ConcavityTest$(\nabla^2 f(U))$ **then**
      NewtonStep$(f, U, \nabla^2 f(U), V, p)$;
      for $i := 1$ **to** p **do**
        if MonotonicityTest$(\nabla f(V^i))$ **then next** i;
        $f_V := f(V^i) \cap$ CenteredForm$(V^i, \nabla f(V^i))$;
        if $\overline{f}(m(V^i)) < \widetilde{f}$ **then** $\widetilde{f} := \overline{f}(m(V^i))$;
          $L_{work} :=$ CutOffTest$(L_{work}, \widetilde{f})$;
        if $\widetilde{f} >= \underline{f_V}$ **then**
          if $w(f_V) < \epsilon$ **then** $L_{res} := L_{res} \cup (V^i, \underline{f_V})$;
            else $L_{work} := L_{work} \cup (V^i, \underline{f_V})$;
  else
    while $(L_{temp} \neq \{\})$ **do**
      $U :=$ Head(L_{temp});
      if $w(f_U) < \epsilon$ **then** $L_{res} := L_{res} \cup (U, \underline{f_U})$;
        else $L_{work} := L_{work} \cup (U, \underline{f_U})$;
  if $L_{work} \neq \{\}$ **then** $Y :=$ Head(L_{work});
until $L = \{\}$;
$Y :=$ Head(L_{res}); $f^* := [\underline{f_Y}, \widetilde{f}]$;
return L_{res}, f^*;

you must issue the `startinlab` command always before using INTLAB, not only the first time.

INTLAB has been successfully tested under different MATLAB versions up to the 7.4 (R2007a) version. In recent releases MATLAB uses Intel Math Kernel Library (IMKL) as default for BLAS operations. This may cause problems, since the IMKL library takes full control over the control word forcing internal computation to be done in extended mode and sets the rounding mode to nearest. In order to make INTLAB work properly, we have to change to another BLAS library, for instance the Atlas library which was used up to recently by MATLAB.

Under Windows, we should change the system variable BLAS_VERSION to atlas***.dll choosing '***' according to the processor type (for example the full name is "atlas_P4.dll" for a PC with a Pentium 4 processor). The corresponding file is located in the MATLAB directory "...\MATLAB\bin\win32\". If you have the rights of the system administrator, then you can set the BLAS version for all users

and threads using the Environment Variables (Control Panel → System Properties → Advanced) dialog box. Otherwise complete the above procedure in the command line window, applying the command

```
set BLAS_VERSION="atlas_P4.dll"
```

and make sure that MATLAB is started then from the same place.

Under Linux we can set the new library by the command

```
export BLAS_VERSION="atlas_P4.so"
```

to reach the above level of readiness. The setting of the BLAS library usually solves all the problems what is reported first by the startintlab procedure.

After starting INTLAB, you may wish to define an interval and see the results of the basic operations and standard functions. You can define an interval with the

```
infsup(a,b)
```

command. INTLAB has the default display mode of intervals with uncertainty, for example

```
infsup(3.14, 3.15)
```

results in

```
intval ans = 3.15_
```

while with the command

```
intvalinit('displayinfsup')
```

the result of the same definition is in the more conventional

```
[3.14, 3.15]
```

form. Now after the definitions `x=infsup(0,1);` and `y=infsup(2,3);` the statement `x/y` provides

```
intval ans = [0.0000, 0.5000]
```

while `sin(x)` gives

```
intval ans = [0.0000, 0.8415].
```

After reaching this point, it is worth to read the article on INTLAB [12], and to study the demo programs.

The new MATLAB/INTLAB based interval global optimization algorithm will also be available soon as a part of the GLOBAL package. The latter is to be downloaded from

`www.inf.u-szeged.hu/∼csendes/reg/regform.php`.

The comprised package contains all necessary files, a suitable directory structure and also a testing environment.

3. Use of the interval global optimization algorithm and examples

After decompressing the archive, move the current directory setting to the new one named GlobalInterval, and within MATLAB, issue the command

```
>> MainTester
```

In case the directory named TestFunctions contains only the files `sh5.bnd` and `sh5.m`, then the interval global optimization algorithm GOP will solve only the Shekel-5 standard global optimization problem.

The content of the file `sh5.bnd` is

```
S5
4
0   10
0   10
0   10
0   10
1e-8
```

The first line provides the name of the function, the second one contains the dimension of the problem (here 4), then the subsequent 4 lines set the coordinate intervals for the search space, which is now a 4-dimensional interval. Finally the last line sets the tolerance ε of the stopping criterion: when the width of the actual inclusion function value is less than this value then the subinterval is moved from the working list to the result list (and the algorithm will not subdivide it further).

The objective function is given in the file `sh5.m`:

```
%Shekel-(4,5)
function y = sh5(x)
%
% Shekel function
% MATLAB Code by A. Hedar (Nov. 23, 2005).
% The number of variables n = 4
% The parameter m should be adjusted m = 5,7,10.
% The default value of m = 10.
%
m = 5;
a = ones(10,4);
a(1,:) = 4.0*a(1,:);
a(2,:) = 1.0*a(2,:);
a(3,:) = 8.0*a(3,:);
a(4,:) = 6.0*a(4,:);
for j = 1:2;
   a(5,2*j-1) = 3.0; a(5,2*j) = 7.0;
   a(6,2*j-1) = 2.0; a(6,2*j) = 9.0;
   a(7,j)     = 5.0; a(7,j+2) = 3.0;
   a(8,2*j-1) = 8.0; a(8,2*j) = 1.0;
   a(9,2*j-1) = 6.0; a(9,2*j) = 2.0;
   a(10,2*j-1)= 7.0; a(10,2*j)= 3.6;
end
c(1) = 0.1; c(2) = 0.2; c(3) = 0.2; c(4) = 0.4; c(5) = 0.4;
c(6) = 0.6; c(7) = 0.3; c(8) = 0.7; c(9) = 0.5; c(10)= 0.5;
s = 0.0;
for j = 1:m;
   p = 0.0;
   for i = 1:4
      p = p+(x(i)-a(j,i))^2;
   end
   s = s+1.0/(p+c(j));
end y = -s;
```

The result we obtain after running MainTester is then:

```
Function name: S5

The set of global minimizers is located in the union of the
following boxes:

c1: [4.00003713662883, 4.00003718945147]
    [4.00013323800906, 4.00013329348396]
    [4.00003713910016, 4.00003717168197]
    [4.00013326916774, 4.00013328566954]

The global minimum is enclosed in:

[-10.153199679058694, -10.153199679058199]

Statistics:

   Iter   Feval   Geval   Heval     MLL  CPUt(sec)
     16     126      86       7      10       6.69
```

In contrast to what can be seen here, it is more usual to have several such result intervals (which are formed from the subintervals of the result list by merging the neighboring ones). The global minimizer points are contained in the union of these. The precision of the inclusion for the global minimum value is much better than the set tolerance. This phenomenon is caused usually by the interval Newton step, which can be very effective on smooth functions.

The optimization algorithm required now 16 iterations, 126 objective function evaluations, 86 gradient evaluations, and 7 Hessian evaluations. The maximal length of the working list was 10, and the CPU time used for the solution was 6.69 seconds. A computational comparison between the introduced new INTLAB based interval global optimization method and the one implemented in C-XSC will be discussed in the next section.

The interval global optimization method can also be applied directly, without MainTester. A simple example for that is to repeat the above test by

```
>> addpath('./','./Utils','TestFunctions')
>> amin=[0; 0; 0; 0]
>> amax=[10; 10; 10; 10]
>> b = infsup(amin,amax)
>> [intv, min, stats] = GOP(@sh5,b,0.00000001)
```

An even simpler usage is given below for the toy problem

$$\min_{x\in[-2,1]^2} x_1^2 + x_2^2 + 1$$

without editing a file that contains the objective function:

```
>> f = inline('x(1)^2+x(2)^2+1')
>> amin = [-2; -2]
>> amax = [1; 1]
>> int =infsup(amin,amax)
>> [intv, min, stat] = GOP(f, int, 0.00000001)
```

Another telling example is the #4 problem from the set of the SIAM 100 $, 100 Digits Challenge announced in 2002 (ten exact digits were to be determined for each of the ten numerical problems). The function to be minimized was:

$$\exp(\sin(50x))+\sin(60e^y)+\sin(70\sin(x))+\sin(\sin(80y))-\sin(10(x+y))+\frac{1}{4}(x^2+y^2).$$

We run our INTLAB based interval global optimization algorithm for the search interval $[-10,10]^2$. The parameter file and the code of the objective function are, respectively:

```
SIAM
2
-10 10
-10 10
1e-8
```

and

```
%SIAM
function y = siam(x)

y = exp(sin(50*x(1))) + sin(60*exp(x(2))) + sin(70*sin(x(1))) +
sin(sin(80*x(2))) - sin(10*(x(1)+x(2))) + 1/4 * (x(1)^2 + x(2)^2);
```

The obtained result was:

```
The set of global minimizers is located in the union of the
following boxes:

c1: [-0.02440308068263, -0.02440307781118]
    [ 0.21061242712377,  0.21061242717589]

The global minimum is enclosed in:

[-3.3068686474752584, -3.3068686474752287]

Statistics:

   Iter   Feval   Geval   Heval     MLL  CPUt(sec)
    238    1723    1151      90      75      75.63
```

That is, we have obtained a very narrow verified enclosure of the global minimum value:

$$[\underline{-3.3068686474752}584, \underline{-3.3068686474752}287]$$

at the cost of 75.63 CPU seconds, 1723 function-, 1151 gradient-, and 90 Hessian inclusion function evaluations, and 238 iterations (using only 75 memory units). The underlined 14 digits are exact. The human overhead (the work to be done by us for coding the problem until the algorithm could be run) of this problem solution was much less than in C-XSC (and Profil/BIAS) – due to the more sophisticated interval standard functions.

Table 1. The numerical comparison of the C-XSC and the INTLAB code. Dim stands for the dimension of the problem, NIT for the number of iterations, NFE for the number of objective function evaluations, and NGE for the number of gradient evaluations.

		Old, C-XSC code			New, INTLAB code		
Problem	Dim	NIT	NFE	NGE	NIT	NFE	NGE
S5	4	16	126	86	16	126	86
S7	4	18	129	84	17	121	78
S10	4	18	126	81	17	123	78
H3	3	42	184	135	42	184	135
H6	6	217	1,014	735	220	1,038	756
GP	2	2,351	15,314	9,430	2,351	15,319	9,427
SHCB	2	130	694	455	130	694	455
THCB	2	56	327	229	56	327	229
BR	2	52	282	200	52	282	200
RB	2	43	263	172	43	263	172
RB5	5	612	4,907	3,685	607	4,878	3,664
L3	2	293	1,890	1,301	293	1,890	1,301
L5	2	88	578	397	88	578	397
L8	3	11	80	55	11	80	55
L9	4	16	112	74	16	115	77
L10	5	19	143	97	19	143	97
L11	8	29	225	152	29	225	152
L12	10	34	282	194	34	282	194
L13	2	12	72	46	12	72	46
L14	3	15	104	66	15	104	66
L15	4	20	135	86	20	135	86
L16	5	19	142	88	19	142	88
L18	7	27	206	130	27	206	130
Schw2.1	2	226	1,312	951	226	1,312	951
Schw3.1	3	14	91	61	14	91	61
Schw2.5	2	53	307	216	53	307	216
Schw2.14	4	369	2,600	1,820	408	2,780	1,913
Schw2.18	2	51	284	201	51	284	201
Schw3.2	3	33	201	135	25	164	110
Schw3.7_5	5	129	677	484	129	677	484
Schw3.7_10	10	7,566	35,385	25,771	7,566	35,385	25,771
Griew5	5	691	9,854	6,424	705	9,940	6,482
Griew7	7	40	272	163	40	272	163
R4	2	154	902	648	153	899	645
R5	3	173	1,555	1,174	173	1,555	1,174
R6	5	227	2,255	1,826	227	2,255	1,826
R7	7	380	4,437	3,711	380	4,437	3,711
R8	9	471	6,170	5,266	471	6,170	5,266
EX2	5	41,794	250,885	177,929	14,774	89,318	65,862

4. Computational tests and comparison

The numerical comparison aimed to clear whether the new implementation is capable to deliver similar quality results as the old one, and to measure the efficiency in terms of the usual indicators. Hence, we have completed a computational test, and compared the efficiency and the results of the INTLAB implementation to that of a C-XSC, BIAS, and Profil based procedure [11].

For the test we used INTLAB version 5.4, MATLAB R2007a, and a PC with 1 Gbyte RAM and a 3 GHz Pentium 4 processor. The test problems included all the standard global optimization functions to be minimized, and basically all of those usually applied in comparing interval global optimization methods. The test function Schwefel 2.7 is missing from the study, and hence also from the tables. The reason for it is that this problem cannot be solved by the algorithms within reasonable time (less than 10 minutes). Otherwise the test problem set is the same as those in other extensive numerical studies, such as [3, 4].

The results are summarized in Tables 1 and 2. The problem names are abbreviated as usual, e.g. S5 stands for Shekel-5, Sch3.2 for Schwefel 3.2, and R4 for Ratz-4 (cf. [3]). The first two columns give the problem names and their dimension. The listed efficiency indicators are the number of iterations necessary (abbreviated as NIT), the number of objective function evaluations (NFE), the number of gradient

Table 2. The numerical comparison of the C-XSC and the INTLAB code. Dim stands for the dimension of the problem, NHE for the number of Hessian evaluations, MLL for the maximal list length required, and CPU for the CPU time needed in seconds.

		Old, C-XSC code			New, INTLAB code		
Problem	dim	NHE	MLL	CPU	NHE	MLL	CPU
S5	4	7	10	0.01	7	10	10.14
S7	4	7	14	0.03	6	14	13.05
S10	4	6	16	0.03	6	17	18.56
H3	3	3	12	0.01	3	12	11.20
H6	6	25	69	0.33	27	69	113.47
GP	2	608	480	0.68	608	480	630.33
SHCB	2	25	51	0.01	25	51	21.06
THCB	2	22	19	0.00	22	19	8.11
BR	2	18	12	0.00	18	12	6.91
RB	2	15	11	0.00	15	11	3.17
RB5	5	411	77	0.45	410	73	220.30
L3	2	97	138	0.16	97	138	115.11
L5	2	28	31	0.04	28	31	41.06
L8	3	5	9	0.00	5	9	4.03
L9	4	6	14	0.01	6	14	7.52
L10	5	8	17	0.02	8	17	11.97
L11	8	10	30	0.10	10	30	29.13
L12	10	12	39	0.24	12	39	46.28
L13	2	3	9	0.00	3	9	2.39
L14	3	5	11	0.00	5	11	4.59
L15	4	6	17	0.01	6	17	7.44
L16	5	6	20	0.01	6	20	9.33
L18	7	8	26	0.05	8	26	18.22
Schw2.1	2	88	26	0.02	88	26	36.78
Schw3.1	3	5	6	0.00	5	6	2.67
Schw2.5	2	28	5	0.00	28	5	4.02
Schw2.14	4	179	78	0.09	190	65	76.53
Schw2.18	2	22	8	0.00	22	8	3.81
Schw3.2	3	12	9	0.00	11	7	3.34
Schw3.7_5	5	32	32	0.03	32	32	25.98
Schw3.7_10	10	1,024	1,024	11.35	1,024	1,024	2,585.11
Griew5	5	597	32	1.04	611	32	575.95
Griew7	7	7	52	0.05	7	52	20.63
R4	2	51	39	0.02	51	39	17.72
R5	3	109	43	0.10	109	43	75.22
R6	5	143	29	0.39	143	29	186.75
R7	7	257	47	1.59	257	47	526.50
R8	9	321	65	3.81	321	65	951.27
EX2	5	19,124	1,969	72.93	6,928	1,610	12,042.27

evaluations (NGE), the number of Hessian evaluations (NHE), the maximal length of the working list (MLL), and the required CPU time in seconds (CPU).

Most of the efficiency indicators have the same or very similar values for the two implementations. We discuss here just the larger and systematic differences. The most significant change is definitely in the CPU time needed: the INTLAB based implementation requires on the average ca. 700 times more time to reach basically the same result. The ratios differ from 165 to 2106, and the median of them is 619. The highest ratio values are related to cases when the CPU time for the C-XSC version were hardly measurably low. It is also worth mentioning, that the lowest ratios belong to those test problems, that required more computation. The reason for this drop in speed is that MATLAB works in interpreter mode, and thus it is no wonder that a machine code program produced by a compiler can reach better times. On the other hand we have to add that we had a well readable, but less optimized coding, and there remained much to improve exploiting the vectorization feature of MATLAB. The bottom line of this comparison is that although the easy use of MATLAB has its price in speed, still for practical problems the Intlab based interval global optimization method can be a useful modeling tool for early phases of optimization projects.

Since the number of iterations, objective function evaluations, gradient calls,

Hessian evaluations and maximal working list lengths are identical for the two algorithms for the majority of test problems, we can certainly conclude that the algorithms are equivalent, and there cannot be significant algorithmic differences. In the remaining cases the slightly changing indicators are caused by the different realizations of the rounding and other hardware depending statements and functions. This finding is also supported by the fact that the somewhat larger differences (ca. 24%, 18%, 18%, 8%, 22%, and ca. 65%, 64%, 63%, 64%, 18%, respectively for the first five indicators in the Tables 1 and 2) obtained for the test problems Schwefel-3.2 and EX2 can well be led back for the flatness of these functions. The better efficiency indicators obtained for the latter cases are in accordance with the fact that the outside rounding necessary for the verified reliable bounds on the range of the functions is more precise in the INTLAB implementation. A smaller part of the CPU time differences is also due to the quicker but less precise interval operations and functions provided by Profil/BIAS.

Summarizing our numerical results, we can state that the computational experiences confirm that the new implementation is in several indicators (e.g. number of function, gradient and Hessian evaluations, number of iterations, and memory complexity) in essence equivalent to that of the old one. The CPU time needed is as a rule by at least two order of magnitude higher for the INTLAB version – as it can be anticipated regarding the interpreter nature of MATLAB. However, further vectorization coding changes in the algorithm and in the objective functions may improve on that. In spite of the lower speed, the new interval global optimization methods can well be suggested as an early modeling and experimentation tool for the verified solution of bound constrained global optimization problems.

Acknowledgements

The present work was supported by the grants Aktion Österreich-Ungarn 60öu6, OTKA T 048377 and T 046822.

References

[1] J. Balogh, T. Csendes, and R.P. Stateva, *Application of a stochastic method to the solution of the phase stability problem: cubic equations of state.* Fluid Phase Equilibria 212 (2003), pp. 257-267.
[2] B. Bánhelyi, T. Csendes, and B.M. Garay, *Optimization and the Miranda approach in detecting horseshoe-type chaos by computer.* Int. J. Bifurcation and Chaos 17 (2007), pp. 735-747.
[3] T. Csendes, *New subinterval selection criteria for interval global optimization.* J. Global Optimization 19 (2001), pp. 307-327.
[4] ———, *Numerical experiences with a new generalized subinterval selection criterion for interval global optimization.* Reliable Computing 9 (2003), pp. 109-125.
[5] T. Csendes, B. Bánhelyi, and L. Hatvani, *Towards a computer-assisted proof for chaos in a forced damped pendulum equation.* J. Computational and Applied Mathematics 199 (2007), pp. 378-383.
[6] T. Csendes, B.M. Garay, and B. Bánhelyi, *A verified optimization technique to locate chaotic regions of Hénon systems.* J. of Global Optimization 35 (2006), pp. 145-160.
[7] T. Csendes et al.; *The GLOBAL Optimization Method Revisited.* Accepted for publication in the Optimization Letters.
[8] R. Hammer et al., *Numerical Toolbox for Verified Computing I.* Springer-Verlag, Berlin, 1993.
[9] R.B. Kearfott, *Rigorous global search: continuous problems.* Kluwer, Dordrecht, 1996.
[10] M.Cs. Markót and T. Csendes, *A new verified optimization technique for the "packing circles in a unit square" problems.* SIAM J. on Optimization 16 (2005), pp. 193-219.
[11] M.C. Markót et al., *New interval methods for constrained global optimization.* Mathematical Programming 106 (2006), pp. 287-318.
[12] S.M. Rump, *INTLAB – Interval Laboratory.* In: T. Csendes (ed.): Developments in Reliable Computing, Kluwer, Dordrecht, 1999, pp. 77-104.
[13] P.G. Szabó et al., *New Approaches to Circle Packing in a Square – With Program Codes.* Springer-Verlag, Berlin, 2007.
[14] B. Tóth, J. Fernández, and T. Csendes, *Empirical convergence speed of inclusion functions for facility location problems.* J. of Computational and Applied Mathematics 199 (2007), pp. 384-389.