Journal of Computations & Modelling, vol.2, no.4, 2012, 149-165
ISSN: 1792-7625 (print), 1792-8850 (online)
Scienpress Ltd, 2012

Scanning solution set of inequality system by a combined code

Ferenc Kálovics[1]

Abstract

Numerical methods often reduce the solving of a complicated problem to a set of elementary problems. In previous papers, the author reduced some problems of numerical analysis (e.g. the computation of integral value with error bound) to computing solution boxes of one inequality. The paper contains a new algorithm for application of solution boxes of one inequality. This algorithm can be useful in itself and can be used as the base of other investigations. Nevertheless, the real worth of the paper is that the first completely automatic, effective code in the topic is given here by connecting a Maple 13 code and a Visual C++ 2008 code.

Mathematics Subject Classification : 65G40, 65D30
Keywords: Inequality system, solution box, combined code

1 Introduction

Illustrate the basic idea of our algorithm on a two-dimensional, drawable

[1] Department of Applied Mathematics, University of Miskolc, H-3515 Miskolc, Hungary, e-mail: matkf@uni-miskolc.hu

Article Info: *Received* : September 25, 2012. *Revised* : October 28, 2012
Published online : December 30, 2012

problem. Let the inequality system

$$f_1(x) = \ln x_1 + |\sin x_1| - x_2 \geq 0, \quad f_2(x) = x_2 - 0.2(x_1 - 7)\,|x_1 - 2| \geq 0$$

be given and suppose that $x = (x_1, x_2) \in \mathit{Dom} = ([1, 11], [-2, 8])$. Try to give a solution box (SB) or a non-solution box (NB) around the point $c = (2, 4)$. Since $f_1(c) < 0$ and $f_2(c) \geq 0$, the continuity of f_1 on *Dom* guarantees an NB around the point $(2, 4)$. Using the addition rule of our program (that comes from [2]) for $f_1(x)$, the sufficient condition

$$\ln x_1 < 1.8919, \quad |\sin x_1| - x_2 < -1.8919$$

is obtained. It is obvious that this is a sufficient condition indeed, furthermore (because of $\ln 2 < 1.8919$ and $|\sin 2| - 4 < -1.8919$) these inequalities can be satisfied around c. Now use the subtraction rule of our program ([2] does not give subtraction rule, it use the identity $a - b = a + (-1)b$ instead of the direct rule) for the second inequality. Then the sufficient condition

$$|\sin x_1| < 1.5087, \quad x_2 > 3.4006$$

is obtained. Naturally these inequalities can also be satisfied around c. If the composite function rule of our program (or [2]) is used for the first inequality, then the evident sufficient condition

$$\sin x_1 > -1.5087, \quad \sin x_1 < 1.5087$$

is obtained. Now the four elementary inequalities

$$\ln x_1 < 1.8919, \quad x_2 > 3.4006, \quad \sin x_1 > -1.5087, \quad \sin x_1 < 1.5087$$

determine an NB in *Dom* around c, i.e. our problem is reduced to simple problems by using three decomposition rules. The evaluation of the elementary inequalities requires only simple mathematical analysis knowledge, and the result is

$$([1, 6.6321], [3.4006, 8]) \subset \mathit{Dom}.$$

The volume (the area) of this NB is 25.9043 units, which is the 25.9043 percentage of the volume of *Dom*. After this lucky choice, try to give an SB or NB around the point $c = (5, 1)$. Since $f_1(c) \geq 0$ and $f_2(c) \geq 0$, the continuity of f_1 and f_2 guarantees an SB around the point $(5, 1)$. Using the addition, subtraction and composite function rules (as 3 decomposition rules) for $f_1(x)$, the elementary inequalities

$$\ln x_1 \geq 0.8253, \quad x_2 \leq 1.3921, \quad \sin x_1 \leq -0.5668$$

are obtained. After evaluation of the elementary inequalities the solution box to $f_1(x) \geq 0$ around c in *Dom* is $([3.7443, 5.6805], [-2, 1.3921])$. Using the subtraction, constant multiplication, multiplication, composite function and constant addition rules (as 5 decomposition rules) for $f_2(x)$, the elementary inequalities

$$x_2 \geq -0.1, \quad x_1 \leq 6.4227, \quad x_1 \leq 7, \quad x_1 \geq 2.8660$$

are obtained. The evaluation is evident here, and the solution box to $f_2(x) \geq 0$ around c in *Dom* is $([2.8660, 6.4227], [-0.1, 8])$. The SB to the original inequality system around c in *Dom* is the intersection of these two boxes, which is

$$([3.7443, 5.6805], [-0.1, 1.3921]) \subset Dom.$$

The volume (the area) of this SB is 2.8891 units, which is the 2.8891 percentage of the volume of *Dom*. (The real numbers rounded to 4 digits came from much more precise values.) For the points $c = (2, 4)$, $c = (5, 1)$, $c = (8, 2)$ and $c = (10, -1)$, SB or NB are illustrated in the Figure 1.

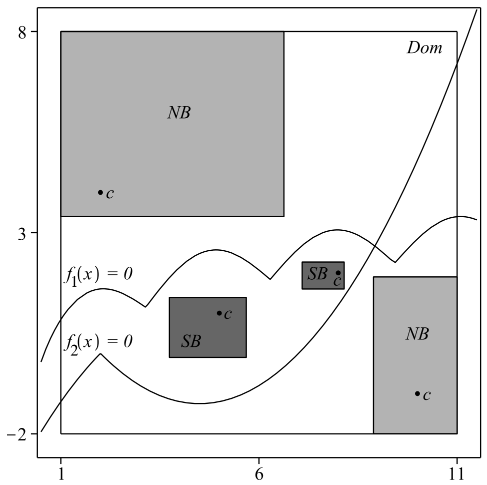

Figure 1: SB or NB around points

The aim of this paper is to give a completely automatic, effective code for scanning the solution set (and the non-solution set) of an inequality system by using the outlined ‘box technics’. Note this box technics (based on [2]) is not a part of the technics used in the so-called interval methods (see e.g. the classic books [1, 6]). The handling and application of these two tools require a highly different mathematical and computational background. The algorithm of Section 3 is a strongly improved version of the algorithm of [5], made suitable for ‘serious scanning’. But the actual problem is the creation of the computer code. The ideal program language for this algorithm should have the expression handling of Maple for using decomposition rules, the speediness of Fortran or C++ in numerical computations for evaluation of elementary inequalities, the numerous standard functions of Fortran and the concise code writing of C++. The programs based on box technics (e.g. in [3, 5]) operate with 3-4 Fortran subroutines or 3-4 C++ function segments and require manual data preparation, i.e. they are not completely automatic codes. The ‘almost ideal’ program of the paper use one Maple 13 procedure for preparation and one connected C++ 2008 function segment for the essential computation. On the other hand, this paper can be considered as a continuation of paper [4].

2 Numerically coded form

Let the (nonlinear) inequality system

$$f_i(x_1, x_2, ..., x_m) \geq 0, \quad x = (x_1, x_2, ..., x_m) \in Dom \subset \mathbb{R}^m, \quad i = 1, 2, ..., n$$

be given. We assume that the multivariate real functions f_i are continuous on the closed box Dom and are built by using the 7 function operations (constant addition ($+c$), constant multiplication ($*c$), addition ($+$), subtraction ($-$), multiplication ($*$), division ($/$), composition of functions ($\circ$)) from the 12 univariate real elementary functions

$$x^r, e^x, \ln x, |x|, \sin x, \cos x, \tan x, \cot x, \arcsin x, \arccos x, \arctan x, \operatorname{arccot} x.$$

It is supposed that r of x^r is a positive integer or negative integer or the reciprocal of a positive integer. This is only a formal restriction because of $x^{k/n} = \left(x^k\right)^{1/n}, \forall k, n \in \mathbb{N}^+$. The functions $a^x, \log_a x, \sinh x$, etc. can be used by the identities $a^x = e^{x \ln a}, \log_a x = \ln x / \ln a, \sinh x = (e^x - e^{-x})/2$, etc., respectively. When the author introduced the decomposition of expressions seen in the introducing example, the process was followed by a Maple program. This code computed a box to one expression very similarly to how it is done in our numerical example. The program utilized to a high extent the ability of Maple to give the type of an expression by the function $type(e, t)$ where e is an expression and t is a type name, furthermore to give operands from an expression by the function $op(i, e)$ where i is a position index and e is an expression. The use of our Maple program was very convenient because we must only give the function (the expression) in a customary form. Unfortunately, the program is rather slow. The conventional program languages (e.g. Fortran, C++) have no similar functions, but they can follow the decomposition if a numerically coded form of the expression (see [2]) used. This form is built of triples of integer or real numbers. The first number in a triple is the so-called operation code (using the order of rules and elementary functions):

$$\begin{aligned}
&+c \Leftrightarrow 1, \quad *c \Leftrightarrow 2, \quad + \Leftrightarrow 3, \quad - \Leftrightarrow 4, \quad * \Leftrightarrow 5, \quad / \Leftrightarrow 6,\\
&x^r \Leftrightarrow 7, \quad \exp \Leftrightarrow 8, \quad \ln \Leftrightarrow 9, \quad \text{abs} \Leftrightarrow 10,\\
&\sin \Leftrightarrow 11, \quad \cos \Leftrightarrow 12, \quad \tan \Leftrightarrow 13, \quad \cot \Leftrightarrow 14,\\
&\arcsin \Leftrightarrow 15, \quad \arccos \Leftrightarrow 16, \quad \arctan \Leftrightarrow 17, \quad \text{arccot} \Leftrightarrow 18.
\end{aligned}$$

The second number in the triples is minus i if an elementary function uses the argument x_i (the minus sign identifies the elementary funcions). If the operation given by the first number of a triple uses a former triple as the first operand, then the ordinal number of this triple is the second number in the actual triple. The third number in the triples is the given constant or the exponent if the constant addition, constant multiplication or the power function is used, respectively. If the operation given by the first number of the triple uses a former triple as a second operand (cases of $+, -, *, /$), then the ordinal number of this triple is the third number in the actual triple. If the operation given by the first number of the triple is $\exp, \ln, ..., \text{arccot}$ (the first number in the triple is between 8 and 18), then the third number in the actual triple is 0. Demonstrate the description on the first expression

$$f_1(x_1, x_2) = \ln x_1 + |\sin x_1| - x_2$$

of our sample problem. The three elementary functions used in the build-up of $f_1(x_1, x_2)$ (the elementary boxes belong to these functions after the decomposition) are $\ln x_1$, $\sin x_1$, and $x_2 = (x_2)^1$. Their descriptions are

$$\ln x_1 = (9, -1, 0), \quad \sin x_1 = (11, -1, 0), \quad x_2 = (7, -2, 1).$$

Now we have the sequence of triples

$$(9,-1,0),(11,-1,0),(7,-2,1).$$

Using the 2nd element of this sequence, the expression $|\sin x_1|$ can be obtained in the form $(10,2,0)$, and this triple is the 4th element of the sequence. Since the last 2 operations ($+$ and $-$) can be written in form $(3,1,4)$ and $(4,5,3)$, the complete description of $f_1(x_1,x_2)$ is:

$$(9,-1,0),(11,-1,0),(7,-2,1),(10,2,0),(3,1,4),(4,5,3).$$

Our Maple 13 procedure *triples* requests the number of functions (nof) and the functions (fs) as input parameters and the output parameter is the two-dimensional array FF. More exactly, $FF[i,1]$ and $FF[i,2]$, where $i=1,...,nof$, inform that the triples of the expression $f_i(x_1,...,x_m)$ are placed from the $FF[i,1]$th row to the $FF[i,2]$th row in FF, i.e. the first nof rows contain index bounds, then the triples of the 1st, 2nd,..., nofth expressions are placed, respectively. The complete Maple 13 code is as follows.

```
> triples:=proc(nof::integer, fs::Array, FF::Array)
  local x,com,tri,ii,v,i,j,k,g,ind,nt,res;
  x:=array(1 .. 10); com:=array(1 .. 10); tri:=array(1 .. 10, 1 .. 3);
  ii:=array(1 .. 10); v:=array(1 .. 3); res:=array(1 .. 10, 1 .. 3);
  # OPERANDS, TRIPLES
  FF[0, 1]:=nof: FF[0, 2]:=nof:
  for j from 1 to nof do
    i:=0: nt:=1: com[1]:=fs[j]:
    while i<nt do
      i:=i+1: g:=com[i]:
      if type(g, name) then v:=[0, op(1, g), 0]: fi:
      if type(g,‘+‘) then
        if type(op(1,g),numeric) then v:=[1,g-op(1,g),op(1,g)]:
        elif type(g-op(1,g),numeric) then v:=[1,op(1,g),g-op(1,g)]:
        else v:=[3,op(1,g),g-op(1,g)]: if op(1,op(2,g))=-1 then v:=[4,op(1,g),op(1,g)-g]: fi: fi: fi:
      if type(g,‘*‘) then
        if type(op(1,g),numeric) then v:=[2,g/op(1,g),op(1,g)]:
        elif type(g/op(1,g),numeric) then v:=[2,op(1,g),u/op(1,g)]:
        else v:=[5,op(1,g),g/op(1,g)]:
          if type(op(2,g),‘^‘) and op(2,op(2,g))<0 then v:=[6,op(1,g),op(2,g)^(-1)]: fi: fi: fi:
      if type(g, ‘^‘) then v:=[7, op(1, g), op(2, g)]: fi:
      if type(g,function) then
        if op(0, g) = exp then v:=[8, op(1, g), 0]: fi:
        if op(0, g) = ln then v:=[9, op(1, g), 0]: fi:
        if op(0, g) = abs then v:=[10, op(1, g), 0]: fi:
        if op(0, g) = sin then v:=[11, op(1, g), 0]: fi:
        if op(0, g) = cos then v:=[12, op(1, g), 0]: fi:
        if op(0, g) = tan then v:=[13, op(1, g), 0]: fi:
```

```
        if op(0, g) = cot then v:=[14, op(1, g), 0]: fi:
        if op(0, g) = arcsin then v:=[15, op(1, g), 0]: fi:
        if op(0, g) = arccos then v:=[16, op(1, g), 0]: fi:
        if op(0, g) = arctan then v:=[17, op(1, g), 0]: fi:
        if op(0, g) = arccot then v:=[18,op(1,g),0]: fi: fi:
      if v[1] = 0 then tri[i,1]:=7: tri[i, 2]:=-v[2]: tri[i, 3]:=1: fi:
      if v[1]>=7 and v[1]<=18 then
        if type(v[2],name) then tri[i,1]:=v[1]: tri[i,2]:=-op(1,v[2]): tri[i,3]:=v[3]:
        else nt:=nt+1: com[nt]:=v[2]: tri[i,1]:=v[1]: tri[i,2]:=nt: tri[i,3]:=v[3]: fi: fi:
      if v[1] = 1 or v[1] = 2 then nt:=nt+1:
        com[nt]:=v[2]: tri[i, 1]:=v[1]: tri[i, 2]:=nt: tri[i, 3]:=v[3]: fi:
      if v[1]>=3 and v[1]<=6 then nt:=nt+1:
        com[nt]:=v[2]: nt:=nt+1: com[nt]:=v[3]: tri[i,1]:=v[1]: tri[i,2]:=nt-1: tri[i,3]:=nt: fi:
    od:
    # NEW INDEXES
    ind := 1:
    for i to nt do if tri[i, 2]<0 then ii[i] := ind: ind := ind+1: fi: od:
    for i from nt by -1 to 1 do if tri[i,2] >0 then ii[i] := ind: ind := ind+1: fi: od:
    # TRIPLE FORM OF THE j-th FUNCTION
    for ind from 1 to nt do
      for i from 1 to nt do
        if ii[i]=ind then res[ind,1]:=tri[i,1]:
          if tri[i,2]<0 then res[ind,2] := tri[i,2]: res[ind,3] := tri[i,3]: fi:
          if tri[i,2]>0 then res[ind,2]:=ii[tri[i,2]]:
            if tri[i,1]>=3 and tri[i,1]<=6 then res[ind,3]:=ii[tri[i,3]]: else res[ind,3]:=tri[i,3]:
        fi: fi: fi:
    od: od:
    FF[j, 1] := FF[j-1, 2]+1: FF[j, 2] := FF[j-1,2]+nt: FF[j, 3] := 0:
    for i to nt do for k to 3 do FF[i+FF[j, 1]-1, k] := res[i, k]: od: od:
  od:
  end proc:
```

Considering the capability of our personal computer, the declarations of arrays in this procedure and in the following C++ function segment suppose that the number of inequalities is less than 10 ($nof < 10$), furthermore every inequality can be coded by at most 30 triples ($FF[i,2] - FF[i,1] < 30$, where $i = 1, 2, ..., nof$). If the procedure is called with the data of our sample problem, then $FF[1,1] = 3$, $FF[1,2] = 8$ and $FF[2,1] = 9$, $FF[2,2] = 17$, i.e. the 6 triples of $f_1(x) = \ln x_1 + |\sin x_1| - x_2$ are placed from the 3rd row to the 8th row in FF and the 9 triples of $f_2(x) = x_2 - 0.2(x_1 - 7)\,|x_1 - 2|$ are placed from the 9th row to the 17th row in FF. The triples are $(9,-1,0)$, $(7,-2,1)$, $(11,-1,0)$, $(10,3,0)$, $(4,4,2)$, $(3,1,5)$ and $(7,-2,1)$, $(7,-1,1)$, $(7,-1,1)$, $(1,3,-2)$, $(10,4,0)$, $(1,2,-7)$, $(5,6,5)$, $(2,7,-0.2)$, $(3,1,8)$. The first result differs from our previous one formally because the procedure does not use the left-right rule, but they are equivalent in essence. The second result has a redundance (the 2nd and 3rd triples are the same), but it does not damage the production. This code shows some similarity with the Maple code of [4], but essential differences are the followings. (1) Paper [4] (and all previous programs) avoided the subtraction rule (because of some Maple difficulties), therefore the coded forms are different there as a matter of course. (2) Paper [4] gives the numerically coded form of one expression only. (3) Paper [4] does not give a procedure, it uses a simple command sequence and the results (the triples) are handled manually.

3 Scanning solution set and non-solution set

Let the (nonlinear) inequality system

$$f_i(x_1, x_2, ..., x_m) \geq 0, \quad x = (x_1, x_2, ..., x_m) \in Dom \subset \mathbb{R}^m, \quad i = 1, 2, ..., n$$

be given (with the conditions seen at the beginning of Section 2). The amplifications are based on the following four principles. (1) Now we have a suitable form of expressions $f_i(x_1, x_2, ..., x_m)$ to make a fast C++ code for the algorithm illustrated in Section 1. (2) The scanning of the solution set S gives an approximation of the volume of S and the scanning of the complementary set $Dom - S$ also facilitates computation of an error bound to the coverage. (3) If U and T are m-dimensional boxes, then the set $U - T$ can be divided into (at most) $2m$ boxes easily. (4) Too small boxes are ignored by the simple condition $vol(B) > bou$, where $vol(B)$ is the volume of B. Naturally, the bound bou has a strong influence on the available error bound. Usually only one property of solution box (SB) or non-solution box (NB) is utilized in computation.

For example, the volume of the box is utilized in computing the integral value with error bound (see [3]) and a vertex of the box is utilized in computing the global minimum of a strictly increasing objective function over Dom. Therefore only a selected set of SBs and NBs is stored for output by the condition $vol(B) > sel$, where $sel \geq bou$. The algorithmic description of the method is as follows.

(a) Define the first element of an interval (box) sequence $\{I_k\}$ by $I_1 = Dom$. Let $nob = 1$, $nsb = 0$, $nnb = 0$, $nign = 0$, $nsel = 0$, $vsb = 0$, $vnb = 0$, $vign = 0$, $vsel = 0$, where nob, nsb, nnb, $nign$, $nsel$, vsb, vnb, $vign$, $vsel$

denote the number of boxes in the sequence $\{I_k\}$, the number of SBs, the number of NBs, the number of ignored boxes, the number of selected boxes, the volume of SBs, the volume of NBs, the volume of ignored boxes, the volume of selected boxes, respectively.

(b) Compute the box SNB (it is SB or NB) around the centre c of I_{nob}.

  (b1) If SNB is SB and $vol(SNB) > bou$ then let $nsb = nsb + 1$, $vsb = vsb + vol(SNB)$.

  (b2) If SNB is NB and $vol(SNB) > bou$ then let $nnb = nnb + 1$, $vnb = vnb + vol(SNB)$.

  (b3) If $vol(SNB) \geq sel$ then let $nsel = nsel{+}1$, $vsel = vsel{+}vol(SNB)$, $Bsel(nsel, 1) = 1$ or -1, $Bsel(nsel, 2) = vol(SNB)$, $Bsel(nsel, 2i{+}1) = SNB(i, 1)$, $Bsel(nsel, 2i{+}2) = SNB(i, 2)$, where $i = 1, 2, ..., m$.

(c) If the set $I_{nob} - SNB$ is empty, then do nothing and let $nb = 0$. If the set $I_{nob} - SNB$ is not empty and $vol(SNB) \leq bou$, then divide the set I_{nob} in two halves along the longest side and let $nb = 2$. If the set $I_{nob} - SNB$ is not empty and $vol(SNB) > bou$, then divide the set $I_{nob} - SNB$ in nb 'almost disjunct' boxes, where $1 \leq nb \leq 2m$. Filter (ignore) the 'unimportant' (too small) boxes by the condition vol(box) $> bou$ and let $nign = nign + 1$, $vign = vign + vol(SNB)$ for every ignoring. Place the $nb^* \leq nb$ new boxes into the box sequence $\{I_k\}$ as nobth, $(nob{+}1)$th,...,$(nob{+}nb^*{-}1)$th elements and let $nob = nob{+}nb^*{-}1$. If $nob > 0$, then go to (b). If $nob = 0$, then load the values nsb, nnb, $nign$, $nsel$, vsb, vnb, $vign$, $vsel$ in the 0th row of the array $Bsel$ and go to the calling point.

C++ code uses the above 'reminding names' and $\{I_k\} \Leftrightarrow Ise$. The essential two cases of the dividing algorithm of (c) for $m = 2$ are illustrated in the Figure 2.

Our C++ 2008 function segment *scanning* requests the number of variables (m), the number of inequalities (n), the domain of inequality system (Dom), the numerically coded form (F) of $f_i(x_1, x_2, ..., x_m)$, $i = 1, 2, ..., n$, the bound bou for ignoring and the bound sel for selecting as input parameters and the output parameter is the two-dimensional array $Bsel$. (It is supposed that $nsel < 1000$.) The complete C++ 2008 code is as follows.

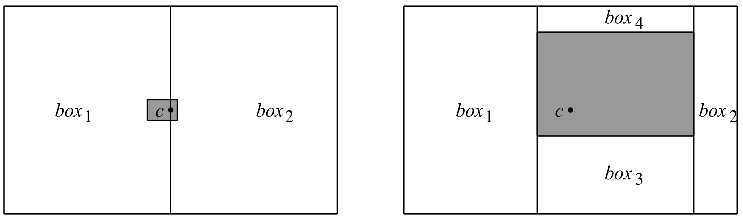

Figure 2: New boxes by dividing algorithm

```
#include "math.h"
double Ise[100000][10][3];
void scanning
(int m,int n,double Dom[10][3],double F[300][4], double bou,double sel,double Bsel[1000][20])
{ double G[30][4],B[10][3],BB[10][3],SNB[10][3],c[10],fv[30], ne[30][5],el[30][5],ax[10][3];
  double vsb,vnb,vign,vsel,vol,vo,al,be,ga,de,ep,x,y,w,alf,per; const double Pi=3.14159265;
  int ind[10],nob,nsb,nnb,nign,nsel,i,j,k,l,ii,jj,kk,nn,nt,ine,iel,s,poi;
  /*INITIAL VALUES*/
  for (i=1; i<=m; i++) {Ise[1][i][1]=Dom[i][1]; Ise[1][i][2]=Dom[i][2];}
  nob=1; nsb=0; nnb=0; nign=0; nsel=0; vsb=0; vnb=0; vign=0; vsel=0;
  while (nob>0)
  { for (i=1; i<=m; i++) {SNB[i][1]=Ise[nob][i][1]; SNB[i][2]=Ise[nob][i][2];
      c[i]=(Ise[nob][i][1]+Ise[nob][i][2])/2; BB[i][1]=c[i]; BB[i][2]=c[i];}; vol=0; poi=1;
    /*SOLUTION BOX OR NON-SOLUTION BOX AROUND A POINT*/
    for (nn=1; nn<=n; nn++) { nt=F[nn][2]-F[nn][1]+1;
      for (i=1; i<=nt; i++) for (ii=1; ii<=3; ii++) G[i][ii]=F[(int)F[nn][1]+i-1][ii];
      /*FUNCTION VALUES TO COMPONENTS OF nn-th FUNCTION*/
      for (i=1; i<=nt; i++)
      { j=G[i][1]; k=G[i][2]; l=G[i][3]; w=G[i][3];
        if (k<0) x=c[-k]; else x=fv[k]; if (j>=3 && j<=6) y=fv[l];
        switch (j)
        { case 1:fv[i]=x+w;break; case 2:fv[i]=x*w;break; case 3:fv[i]=x+y;break;
          case 4:fv[i]=x-y;break; case 5:fv[i]=x*y;break; case 6:fv[i]=x/y;break;
          case 7:fv[i]=pow(x,w);break; case 8:fv[i]=exp(x);break; case 9:fv[i]=log(x);break;
          case 10:fv[i]=fabs(x);break; case 11:fv[i]=sin(x);break; case 12:fv[i]=cos(x);break;
          case 13:fv[i]=tan(x);break; case 14:fv[i]=1/tan(x);break; case 15:fv[i]=asin(x);break;
          case 16:fv[i]=acos(x);break; case 17:fv[i]=atan(x);break;
          case 18:fv[i]=Pi/2-atan(x);break;}}
      if (fv[nt]<0) poi=-1;
      /*DECOMPOSITION BY USING THE 7 RULES*/
      for (i=1; i<=3; i++) ne[1][i]=G[nt][i]; ne[1][4]=0; ine=1; iel=0; i=0;
      while (i<ine && nt>1)
      { i++; j=ne[i][1]; k=ne[i][2]; l=ne[i][3]; w=ne[i][3]; al=ne[i][4]; s=0;
        switch (j)
        { case 1:be=al-w;s=1;break;
          case 2:be=al/w;s=1;break;
          case 3:be=(al+fv[k]-fv[l])/2;ga=al-be;s=2;break;
          case 4:be=(al+fv[k]+fv[l])/2;ga=be-al;s=2;break;
          case 5: if (al==0||fv[k]*fv[l]*al<0) {be=0;ga=0;s=2;}
            if (al!=0 && fv[k]==0 && fv[l]==0) {be=sqrt(fabs(al));ga=be;de=-be;ep=de;s=4;}
            if (al!=0 && fv[k] !=0 && fv[l]==0) {be=2*fv[k];ga=al/be;de=0;s=3;}
            if (al!=0 && fv[k]==0 && fv[l]!=0) {ga=2*fv[l];be=al/ga;de=be;ep=0;s=4;}
            if (fv[k]*fv[l]*al>0) {be=fv[k]/fabs(fv[k])*sqrt(al*fv[k]/fv[l]);
              ga=fv[l]/fabs(fv[l])*sqrt(al*fv[l]/fv[k]);de=0;s=3;};break;
          case 6: if (al==0) {be=0;s=1;}
            if (al !=0) {ga=(al*fv[k]+fv[l])/(1+al*al);be=al*ga;s=2;};break;
          case 7: if (w>0 && (l+1)/2*2==l+1) {be=pow(al,1/w); s=1;}
            if (w>0 && l/2*2==l && al>=0) {be=pow(al,1/w);ga=-be;s=2;}
            if (w<0 && (l-1)/2*2==l-1 && al !=0) {be=pow(al,1/w);s=1;}
            if (w<0 && l/2*2==l && al>0) {be=pow(al,1/w);ga=-be;s=2;}
            if (w>0 && w<1 && al>=0) {be=pow(al,1/w);s=1;};break;
          case 8: if (al>0) {be=log(al);s=1;};break;
          case 9:be=exp(al);s=1;break;
          case 10: if (al>= 0) {be=al;ga=-al;s=2;};break;
          case 11:
            if (fabs(al)<=1) {x=asin(fabs(al));per=2*Pi*(int)(fv[k]/2/Pi);
            if(fv[k]<0) per=per-2*Pi; if (al>=0) {be=x+per;ga=Pi-x+per;}
            if (al<0) {be=Pi+x+per;ga=2*Pi-x+per;}; if (ga<fv[k]) {y=be;be=ga;ga=y+2*Pi;}
            if (be>fv[k]) {y=ga;ga=be;be=y-2*Pi;}; s=2;};break;
          case 12:
            if (fabs(al)<=1) {x=acos(fabs(al));per=2*Pi*(int)(fv[k]/2/Pi);
            if(fv[k]<0) per=per-2*Pi; if (al>=0) {be=x+per;ga=2*Pi-x+per;}
            if (al<0) {be=Pi-x+per;ga=Pi+x+per;}; if (ga<fv[k]) {y=be;be=ga;ga=y+2*Pi;}
            if (be>fv[k]) {y=ga;ga=be;be=y-2*Pi;}; s=2;};break;
          case 13:x=atan(fabs(al)); per=2*Pi*(int)(fv[k]/2/Pi); if(fv[k]<0) per=per-2*Pi;
            if (al>=0) {be=x+per;ga=Pi+x+per;}; if (al<0) {be=Pi-x+per;ga=2*Pi-x+per;}
            if (ga<fv[k]) {be=ga;ga=be+Pi;}; if (be>fv[k]) {ga=be;be=ga-Pi;};s=2;break;
          case 14:x=Pi/2-atan(fabs(al)); per=2*Pi*(int)(fv[k]/2/Pi); if(fv[k]<0) per=per-2*Pi;
            if (al>=0) {be=x+per;ga=Pi+x+per;}; if (al<0) {be=Pi-x+per;ga=2*Pi-x+per;}
            if (ga<fv[k]) {be=ga;ga=be+Pi;}; if (be>fv[k]) {ga=be;be=ga-Pi;};s=2;break;
          case 15: if (fabs(al)<=Pi/2) {be=sin(al);s=1;};break;
          case 16: if (al>=0 && al<=Pi) {be=cos(al);s=1;};break;
          case 17: if (fabs(al)<Pi/2) {be=tan(al);s=1;};break;
          case 18: if (al>0 && al<Pi) {be=1/tan(al);s=1;};break;}
        for (ii=1; ii<=s; ii++)
        { switch (ii)
          { case 1:alf=be;kk=k;break; case 2: alf=ga; if (j>=3 && j<=6) kk=l;break;
            case 3:alf=de;kk=k;break; case 4: alf=ep; if (j>=3 && j<=6) kk=l;break;}
          if (G[kk][2]<0) {iel++; for (jj=1; jj<=3; jj++) el[iel][jj]=G[kk][jj];el[iel][4]=alf;}
          if (G[kk][2]>0) {ine++; for (jj=1; jj<=3; jj++) ne[ine][jj]=G[kk][jj];ne[ine][4]=alf;}}}
      if (nt==1) {for (ii=1; ii<=4; ii++) el[1][ii]=ne[1][ii];iel=1;}
      /*EVALUATION OF THE ELEMENTARY INEQUALITIES, 12 CASES*/
      for (i=1; i<=m; i++) {B[i][1]=Ise[nob][i][1]; B[i][2]=Ise[nob][i][2];}
      for (i=1; i<=iel; i++)
      { j=el[i][1]; k=-el[i][2]; l=el[i][3]; w=el[i][3]; al=el[i][4]; s=0;
        switch (j)
        { case 7: if (w>0 && (l+1)/2*2==l+1) {be=pow(al,1/w);s=1;}
            if (w>0 && l/2*2==l && al>=0) {be=pow(al,1/w);ga=-be;s=2;}
            if (w<0 && (l-1)/2*2==l-1 && al!=0) {be=pow(al,1/w);s=1;}
            if (w<0 && l/2*2==l && al>0) {be=pow(al,1/w);ga=-be;s=2;}
            if (w>0 && w<1 && al>=0) {be=pow(al,1/w);s=1;};break;
          case 8: if (al>0) {be=log(al);s=1;};break;
          case 9:be=exp(al);s=1;break;
          case 10:if (al>= 0) {be=al;ga=-al;s=2;};break;
          case 11:
            if (fabs(al)<=1) {x=asin(fabs(al));per=2*Pi*(int)(c[k]/2/Pi);
            if(c[k]<0) per=per-2*Pi; if (al>=0) {be=x+per;ga=Pi-x+per;}
            if (al<0) {be=Pi+x+per;ga=2*Pi-x+per;}; if (ga<c[k]) {y=be;be=ga;ga=y+2*Pi;}
            if (be>c[k]) {y=ga;ga=be;be=y-2*Pi;}; s=2;};break;
          case 12:
            if (fabs(al)<=1) {x=acos(fabs(al)); per=2*Pi*(int)(c[k]/2/Pi);
            if(c[k]<0) per=per-2*Pi; if (al>=0) {be=x+per;ga=2*Pi-x+per;}
            if (al<0) {be=Pi-x+per;ga=Pi+x+per;}; if (ga<c[k]) {y=be;be=ga;ga=y+2*Pi;}
            if (be>c[k]) {y=ga;ga=be;be=y-2*Pi;}; s=2;};break;
          case 13:x=atan(fabs(al)); per=2*Pi*(int)(c[k]/2/Pi); if(c[k]<0) per=per-2*Pi;
            if (al>=0) {be=x+per;ga=Pi+x+per;}; if (al<0) {be=Pi-x+per;ga=2*Pi-x+per;}
            if (ga<c[k]) {be=ga;ga=be+Pi;}; if (be>c[k]) {ga=be;be=ga-Pi;};s=2;break;
          case 14:x=Pi/2-atan(fabs(al)); per=2*Pi*(int)(c[k]/2/Pi); if(c[k]<0) per=per-2*Pi;
            if (al>=0) {be=x+per;ga=Pi+x+per;}; if (al<0) {be=Pi-x+per;ga=2*Pi-x+per;}
            if (ga<c[k]) {be=ga;ga=be+Pi;}; if (be>c[k]) {ga=be;be=ga-Pi;};s=2;break;
          case 15: if (fabs(al)<=Pi/2) {be=sin(al);s=1;};break;
          case 16: if (al>=0 && al<=Pi) {be=cos(al);s=1;};break;
          case 17: if (fabs(al)<Pi/2) {be=tan(al);s=1;};break;
          case 18: if (al>0 && al<Pi) {be=1/tan(al);s=1;};break;}
        for (ii=1; ii<=s; ii++) { alf=be; if (ii==2) alf=ga;
          if (alf<c[k] && alf>B[k][1]) B[k][1]=alf; if (alf>c[k] && alf<B[k][2]) B[k][2]=alf;}}
      /*SOLUTION BOX OR NON-SOLUTION BOX FOR THE nn-th FUNCTION*/
      if (poi==1)
      { for (i=1; i<=m; i++) {if (B[i][1]>SNB[i][1]) SNB[i][1]=B[i][1];
        if (B[i][2]<SNB[i][2]) SNB[i][2]=B[i][2];}}
      if (fv[nt]<0)
      { vo=1; for (i=1; i<=m; i++) vo=vo*(B[i][2]-B[i][1]);
        if (vo>vol) {vol=vo; for (i=1; i<=m; i++) {BB[i][1]=B[i][1];BB[i][2]=B[i][2];}}}}
    /*THE RESULT: 1 OR -1, VOLUME, SOLUTION BOX OR NON-SOLUTION BOX*/
    if (poi==1) {SNB[0][1]=poi; vo=1;
      for (i=1; i<=m; i++) vo=vo*(SNB[i][2]-SNB[i][1]); SNB[0][2]=vo;}
    if (poi==-1) {SNB[0][1]=poi;
      for (i=1; i<=m; i++) {SNB[i][1]=BB[i][1];SNB[i][2]=BB[i][2];};SNB[0][2]=vol;}
    /*HANDLING RESULT: SERIAL NUMBERS, VOLUMES, LARGE BOXES*/
    if (SNB[0][1]==1 && SNB[0][2]>=bou) {nsb=nsb+1; vsb=vsb+SNB[0][2];}
    if (SNB[0][1]==-1 && SNB[0][2]>=bou) {nnb=nnb+1; vnb=vnb+SNB[0][2];}
    if (SNB[0][2]>=sel)
    { nsel=nsel+1; vsel=vsel+SNB[0][2]; Bsel[nsel][1]=SNB[0][1]; Bsel[nsel][2]=SNB[0][2];
      for (i=1; i<=m; i++) {Bsel[nsel][2*i+1]=SNB[i][1]; Bsel[nsel][2*i+2]=SNB[i][2];}}
    /*NEW BOXES: DIVIDING ALGORITHM*/
    for (i=1; i<=m; i++) {ax[i][1]=Ise[nob][i][1]; ax[i][2]=Ise[nob][i][2];}; nob=nob-1;
    for (i=1; i<=m; i++) {j=1;
      for (k=1; k<=m; k++) if (ax[k][2]-ax[k][1]>ax[i][2]-ax[i][1]) j=j+1;
      for (k=1; k<=m; k++) if (ax[k][2]-ax[k][1]==ax[i][2]-ax[i][1] && k<i) j=j+1; ind[j]=i;}
    if (SNB[0][2]<bou) {for (i=1; i<=m; i++) {SNB[i][1]=ax[i][1]; SNB[i][2]=ax[i][2];};
      SNB[ind[1]][1]=c[ind[1]]; SNB[ind[1]][2]=c[ind[1]];}
    for (i=1; i<=m; i++) {j=ind[i];
      if (ax[j][1]<SNB[j][1]) {nob=nob+1; for (k=1; k<=m; k++)
        {Ise[nob][k][1]=ax[k][1]; Ise[nob][k][2]=ax[k][2];}; Ise[nob][j][2]=SNB[j][1];
        ax[j][1]=SNB[j][1]; vo=1; for (k=1; k<=m; k++) vo=vo*(Ise[nob][k][2]-Ise[nob][k][1]);
        if (vo<bou) {nob=nob-1; nign=nign+1; vign=vign+vo;}}
      if (ax[j][2]>SNB[j][2]) {nob=nob+1; for (k=1; k<=m; k++)
        {Ise[nob][k][1]=ax[k][1]; Ise[nob][k][2]=ax[k][2];}; Ise[nob][j][1]=SNB[j][2];
        ax[j][2]=SNB[j][2]; vo=1; for (k=1; k<=m; k++) vo=vo*(Ise[nob][k][2]-Ise[nob][k][1]);
        if (vo<bou) {nob=nob-1; nign=nign+1; vign=vign+vo;}}}}
    /*FINAL RESULT: NUMBERS OF DIFFERENT BOXES, VOLUMES, SELECTED BOXES*/
    Bsel[0][1]=nsb; Bsel[0][2]=nnb; Bsel[0][3]=nign; Bsel[0][4]=nsel;
    Bsel[0][5]=vsb; Bsel[0][6]=vnb; Bsel[0][7]=vign; Bsel[0][8]=vsel;}
```

It is practical that the large array Ise is defined outside the function segment *scanning*. Since the equality $vsb + vnb + vign = vol(Dom)$ has to be valid, we have a simple 'control' for the correct computation. This code shows some similarity with previous Fortran and C++ codes of the author, but fundamental differences are the followings. (1) All previous programs avoided the subtraction rule, therefore the coded forms are different there as a matter of course. (2) Previous programs use a separated segment for computing function values from numerically coded forms. This segment is eliminated from our code by a single new command. (3) Previous programs use a separated segment for dividing algorithm. A simplified and modified version of this segment is built in here. (4) The parameter list here, which produces connection to the Maple program, is very different from the previous ones.

If the file *triples.m* is made from the file *triples.mpl* with the command

$$save\ triples, "triples.m"$$

and the file *scanning.dll* is made from the file *scanning.cpp* with the command

$$cl\ -Gz\ scanning.cpp\ -link\ -dll\ -export: scanning\ -out: scanning.dll$$

then the two processes can be called very easily from a Maple 13 work-sheet. The calling (main) segment with the inequality system of our sample problem (using the very modest values $bou = 0.3$, $sel = 0.3$) is as follows.

```
> read "triples.m":
> nof:=2:
> fs:=Array([ln(x[1])+abs(sin(x[1]))-x[2], x[2]-(.2*(x[1]-7))*abs(x[1]-2)],datatype = algebraic):
> FF:=Array(0 .. 299, 0 .. 3): fs:=convert(fs, rational):
> triples(nof, fs, FF):
> Mscanning := define_external('scanning', m0::(integer[4]), n0::(integer[4]),
  Dom0::(ARRAY(0 .. 9, 0 .. 2, float[8])), F0::(ARRAY(0 .. 299, 0 .. 3, float[8])),
  bou0::(float[8]), sel0::(float[8]), Bsel0::(ARRAY(0.. 99, 0 .. 19, float[8])), LIB = "scanning.dll"):
> Dom:=Array(0 .. 9, 0 .. 2, datatype = float[8], order = C_order):
> F:=Array(0 .. 299, 0 .. 3, datatype = float[8], order = C_order):
> Bsel:=Array(0 .. 99, 0 .. 19, datatype = float[8], order = C_order):
> m:=2: n:=2: Dom[1, 1]:=1: Dom[1, 2]:=11: Dom[2, 1]:=-2: Dom[2, 2]:=8:
> for i to FF[n, 2] do for k to 3 do F[i, k]:=FF[i, k]: od: od: bou:=0.3: sel:=0.3:
> Mscanning(m, n, Dom, F, bou, sel, Bsel):
```

```
> printf("Box numbers: %.0f %.0f %.0f %.0f \n", Bsel[0, 1], Bsel[0, 2], Bsel[0, 3], Bsel[0, 4]);
> printf("Volumes: %.6f %.6f %.6f %.6f \n", Bsel[0, 5], Bsel[0, 6], Bsel[0, 7], Bsel[0, 8]);
> for i to Bsel[0, 4] do printf("Box: %.0f %.4f %.2f %.2f %.2f %.2f \n",
  Bsel[i, 1], Bsel[i, 2], Bsel[i, 3], Bsel[i, 4], Bsel[i, 5], Bsel[i, 6]): od:
Box numbers:  14 29 98 43
Volumes:  10.065621 71.029243 18.905136 81.094864
Data of 43 boxes are in the following 43 rows.
```

The commands of the first 5 rows activate the procedure *triples* and the commands of the next 9 rows activate the function *scanning*. The commands of the last 4 rows provide the printout of the result. Hence 14 solution boxes (with volume 10.065621), 29 non-solution boxes (with volume 71.029243), 98 ignored boxes (with volume 18.905136) and 43 $(14 + 29)$ selected boxes (with volume 81.094864) are produced and $10.065621+71.029243+18.905136 = 100$. This modest scanning is illustrated in Figure 3.

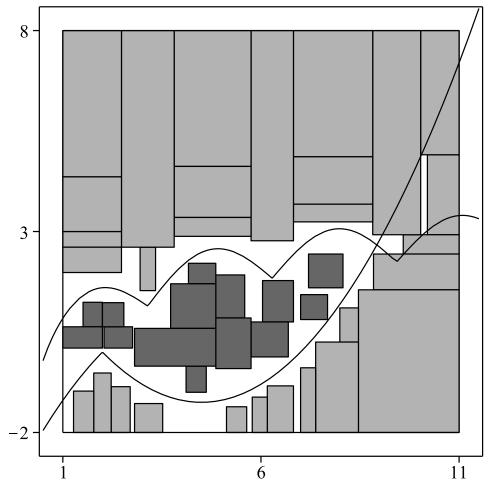

Figure 3: A scanning of 81.094864%

If the values $bou = 10^{-4}$, $sel = 5$ are used in the calling segment, then the result is:

```
Box numbers:  1810 1787 5339 7
Volumes:  19.263489 80.395691 0.340820 46.179833

Data of 7 boxes are in the following 7 rows.
```

Since the sum of the first 3 volumes is 100, we have a scanning of $(100 - 0.340820)\% = 99.659180\%$. If the values $bou = 10^{-6}$, $sel = 5$ are used in the calling segment, then the result is:

```
Box numbers:  17729 17839 54312 7
Volumes:  19.416344 80.549397 0.034259 46.179833
```

```
Data of former 7 boxes are in the following 7 rows.
```

Since the sum of the first 3 volumes is 100, we have a scanning of $(100 - 0.034259)\% = 99.965741\%$. The 7 largest (non-solution) boxes give a covering of 46.179833% in both cases.

For a comparison of running times, choose a more demanding problem. Try to give a good approximation value with guaranted error bound for the integral value

$$\int\int\int_V (x_2^2 + x_3^2) dx_1 dx_2 dx_3,$$

where V is described by the inequality

$$(2 - x_3)^2 - 4x_1^2 - 4x_2^2 \geq 0, \quad x_3 \in [0, 2].$$

The computation of a triple integral for a cone region (the radius of the base circle is 1 unit, the altitude is 2 units) is the problem now. The volume of the solution set of the inequality system

$$\begin{aligned}
&(2 - x_3)^2 - 4x_1^2 - 4x_2^2 \geq 0\\
&x_2^2 + x_3^2 - x_4 \geq 0\\
&x = (x_1, x_2, x_3, x_4) \in \mathit{Dom} = ([-1, 1], [-1, 1], [0, 2], [0, 5])
\end{aligned}$$

gives the required value (consider the geometrical meaning of simple and double integrals). The input data $\mathit{nof} = 2$, $f_1(x) = (2 - x_3)^2 - 4x_1^2 - 4x_2^2$ and $f_2(x) = x_2^2 + x_3^2 - x_4$ to Maple 13 procedure *triples* need to be placed in the 2nd and 3rd rows of the calling segment and the input data $m = 4$, $n = 2$, $\mathit{Dom} = ([-1, 1], [-1, 1], [0, 2], [0, 5])$ to C++ 2008 function segment *scanning* need to be placed in the 12th row of the calling segment. If the values $\mathit{bou} = 10^{-6}$, $\mathit{sel} = 40 = \mathit{vol}(\mathit{Dom})$ are used in the 13th row of the calling segment, then the result is:

```
Box numbers:  51191 57314 390993 0
Volumes:  1.024918 38.726149 0.248933 0.000000

There are not selected boxes here.
```

Since the sum of the first 3 volumes is 40, the value 1.024918 is a guaranted lower bound and $1.024918 + 0.248933 = 1.273851$ is a guaranted upper bound to the integral value. Furthermore the value $1.024918 + 0.248933 * 0.5 = 1.149385$ is an approximating value to the integral value and $0.248933 * 0.5 = 0.124467$ is a guaranted error bound to this value. The exact value of the triple integral (which can be obtained by using cylinder coordinates) is $11\pi/30 \approx 1.151917$, thus the real error of the approximating value is only 0.002532. The result comes from a scanning of 99.377667%, the running time (on our PC of two 2.2 GHz processors) is 2.3 sec . If the function segment *scanning* is called from a C++ main segment (triples are handled manually), i.e. *exe* file is used instead of *dll* file, then the above result comes after 2.4 sec. The code based on the algorithm of [5] is unsuitable for such a computation. The original code based on the algorithm of [3] contains five Fortran 90 segments and uses some 'special tricks', among others the above correction by utilization of the lower and upper bounds. The input parameters are m, n, Dom, F (triples are used manually) and Eb (an upper bound for number of examined boxes). The C++ 'mirror version' of this code with $Eb = 60000$ gives the result:

```
Integral value, error bound:  1.154199  0.210647.
```

The real error is 0.002282, the result comes from a scanning of 98.946765% and the running time (on our PC of two 2.2 GHz processors) is 3.2 sec . The application of our connected code is convenient as using one Maple segment (numerically coded form of inequality system does not appear) and the running time is as good as that of a complete C++ code, i.e. our aim is fulfilled. The author would gladly send the Maple and C++ codes of the paper to interested readers in e-mail as an attached file.

ACKNOWLEDGEMENTS. This research was carried out as part of the TAMOP-4.2.1.B-10/2/KONV-2010-0001 project with support by the European Union, co-financed by the European Social Fund.

References

[1] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *Numerical Toolbox for Verified Computing I.*, Springer-Verlag, Berlin, 1993.

[2] F. Kálovics, Creating and Handling Box Valued Functions Used in Numerical Methods, *J. Comput. Appl. Math.*, **147**, (2002), 333–348.

[3] F. Kálovics, Zones and Integrals, *J. Comput. Appl. Math.*, **182**, (2005), 243–251.

[4] F. Kálovics, A New Tool: Solution Boxes of Inequality, *J. Softw. Eng. Appl.*, **3**, (2010), 737–745.

[5] F. Kálovics and G. Mészáros, Box Valued Functions in Solving Systems of Equations and Inequalities, *Numer. Algorithms*, **36**, (2004), 1–12.

[6] R.B. Kearfott, *Rigorous Global Search: Continuous Problems*, Kluwer Academic Publishers, Dordrecht, 1996.