Miskolc Mathematical Notes
Vol. 8 (2007), No 2, pp. 169-179

HU e-ISSN 1787-2413
DOI: 10.18514/MMN.2007.179

Two improved zone methods

F. Kálovics

Miskolc Mathematical Notes
Vol. 8 (2007), No. 2, pp. 169–179

HU e-ISSN 1787-2413

TWO IMPROVED ZONE METHODS

F. KÁLOVICS

Received 11 August, 2007

Abstract. The first section gives a very simple introduction to zone functions. Using this function, the next two sections present algorithms for computing integral values and global maxima with error bounds. These algorithms are simpler and more effective than the previous ones published by the author. The last section describes some experience about the C++ and Fortran codes used.

1991 *Mathematics Subject Classification:* 65D30, 65K05

Keywords: zone function, definite integral, global maximum

1. INTRODUCTION

Computational methods very often reduce the solving of a complicated task to a set of elementary problems. Since the finding of solution boxes of a system of inequalities, the computation of integral values with error bounds, the approximation of global maxima and others can be reduced to computing solution boxes of one inequality, therefore the creation and the handling by computer code of these solution boxes (as new tools of computational methods) are useful issues. First consider a particular case. Let

$$g : D \subset \mathbb{R}^2 \to \mathbb{R}, \quad g(x_1, x_2) = \sin x_1 + \frac{x_1^2}{\ln x_2}, \text{ where } D = ((0, \pi), (1, 10)).$$

Try to give a solution box to the inequality $g(x_1, x_2) > \alpha$ around the point $c = (1, 2) \in D$ for $\alpha = 1$. Since $g(c) > 1$ and g is continuous on D, there are solution boxes in D around c. It comes from the structure of g that

$$\sin x_1 > 0.5, \quad \frac{x_1^2}{\ln x_2} > 0.5$$

is a sufficient condition for satisfying $g(x_1, x_2) > 1$. Since $x_1^2 > 0.5$ and $\ln x_2 < 1$ is a sufficient condition for satisfying $x_1^2 / \ln x_2 > 0.5$, therefore the three elementary inequalities

$$\sin x_1 > 0.5, \quad x_1^2 > 0.5, \quad \ln x_2 < 1$$

determine the solution box (zone)

$$Z_g(c,\alpha) = \Big(\big(\max(\pi/6, \sqrt{0.5}), \min(5\pi/6, \pi)\big), (1, e)\Big) \approx ((0.71, 2.62), (1, 2.72)).$$

Fortunately, $c \in Z_g(c,\alpha)$ is satisfied, i. e., our casual choices in producing the three elementary inequalities were suitable. Notice that the inequality $g(x_1, x_2) > \alpha$ is equivalent to $g(x_1, x_2) \neq \alpha$ because of the continuity, hence the cases $<, >$ can be discussed at the same time. Now let

$$g : D \subset \mathbb{R}^m \to \mathbb{R}, \text{ where } D = \big((\underline{x}_1, \overline{x}_1), (\underline{x}_2, \overline{x}_2), \ldots, (\underline{x}_m, \overline{x}_m)\big)$$

be such a continuous (multivariate real) function on the open box D that is built of the well-known (univariate real) elementary functions

$$x^a, a^x, \log_a x, |x|, \sin x, \ldots, \arcsin x, \ldots, \sinh x, \ldots, \sinh^{-1} x, \ldots$$

by function operations $+$, $*$, $\circ$ (the last symbol being used for composite functions). The zone function

$$(c,\alpha) \mapsto Z_g(c,\alpha), \text{ where } c \in D, \quad g(c) \neq \alpha,$$

assigns a nonempty open box (interval, zone) $Z_g(c,\alpha) \subset D$ around point c to every pair (c,α), in which the function value $g(x)$ is not equal to α anywhere. If $g(c) < \alpha$, then, because of the continuity of g on $Z_g(c,\alpha)$,it is also true that $x \in Z_g(c,\alpha)$ implies $-\infty < g(x) < \alpha$. Consequently, the zone function Z_g assigns the box $Z_g(c,\alpha)$ of the domain to the interval $(-\infty, \alpha)$ of function values. Similarly, if $g(c) > \alpha$, then the zone function Z_g assigns the box $Z_g(c,\alpha)$ of the domain to the interval (α, ∞) of function values. This property was used intensively in finding solution boxes of systems of inequalities [5], in computing integral values [4], and in approximating global minima [3]. Here we note that interval extension functions used in interval methods (see, e. g., [1, 6]) are inverse type functions, they assign intervals of function values to boxes of domain. The handling of zone functions and interval extension functions requires a highly different computational background. The creation of zone functions is based on two facts:

- zone functions to the above mentioned univariate elementary functions can be created easily,
- the creation of a zone function of a multivariate function g can be reduced to the previous case, by rules belonging to the function operations $+$, $*$, $\circ$ (by suitable choices in every step of the reduction).

Naturally a zone function is not given by a formula, but its values (boxes, zones) are defined uniquely by reduction rules. A variety of rules can be seen in [2]. A reduction seen in our particular example can be easily followed by a Maple code, because Maple can recognize the operands of an expression. Unfortunately, the Maple programs are slow, therefore the author uses a simple numerically coded form of g in

C++ and Fortran programs nowadays. The experience with Maple V Release 5, Visual C++ version 6.0 and Lahey Fortran 90 version 4.5 codes used in [2] can be summarized as follows:

- Maple requires only the conventional (convenient) form of g, C++ and Fortran require a numerically coded form of g,
- the computation efforts (the evaluation times) belonging to $Z_g(c,\alpha)$ and $g(c)$ can be characterized by the formula: $\text{effort}\big(Z_g(c,\alpha)\big) \approx 10 * \text{effort}(g(c))$, with respect to all three programming languages,
- the speeds belonging to the evaluation of $Z_g(c,\alpha)$ satisfy the relations: $\text{speed}(\text{C++}) \approx 200 * \text{speed}(\text{Maple})$, $\text{speed}(\text{Fortran}) \approx 300 * \text{speed}(\text{Maple})$.

At the end of this section, let us emphasize some facts about zone functions:

(i) The reduction process on the expression $g(x)$, the determination of the elementary inequalities, uses only the structure of the expression $g(x)$ and supposes only the continuity of g on D.

(ii) The zone $Z_g(c,\alpha)$ is not a symmetrical box around c. Often it has a large volume, although $g(c) \neq \alpha$ is only just satisfied.

(iii) The preparation of a numerically coded form of the expression $g(x)$ is easy (see in [2]), nevertheless the author also has a short Maple program for this work.

2. Algorithm for computing integrals

Let the definite integral

$$\int \cdots \int_V f(x_1, x_2, \ldots, x_{m-1}) dx_1 dx_2 \ldots dx_{m-1}$$

be given, where the $m-1$ dimensional point set V is described by the system of inequalities

$$f_i(x_1, x_2, \ldots, x_{m-1}) \geq 0, \quad i = 1, 2, \ldots, n-1,\ m \geq 2,\ n \geq 1,$$

the multivariate real functions $f, f_1, f_2, \ldots, f_{n-1}$ are continuous on the closed box $D \supset V$ and are built from the above univariate real elementary functions by using the usual function operations. Let us assume that we know (rough) lower and upper bounds $\underline{x}_m \leq 0$, $\overline{x}_m \geq 0$ so that

$$\underline{x}_m \leq f(x_1, x_2, \ldots, x_{m-1}) \leq \overline{x}_m, \quad \forall (x_1, x_2, \ldots, x_{m-1}) \in D.$$

According to [4] the computation of the integral value is equivalent (consider the geometrical meaning of definite integrals) to the computation of the volumes of the solution sets of the two systems of inequalities

$$\begin{aligned} f_i(x_1, x_2, \ldots, x_{m-1}) &\geq 0, \quad i = 1, 2, \ldots, n-1, \\ f(x_1, x_2, \ldots, x_{m-1}) - x_m &\geq 0, \end{aligned} \tag{2.1}$$

where $(x_1,\dots,x_m)\in I_+ = D\times[0,\overline{x}_m] = \big([\underline{x}_1,\overline{x}_1],\dots,[\underline{x}_{m-1},\overline{x}_{m-1}],[0,\overline{x}_m]\big)$, and

$$\begin{aligned} f_i(x_1,x_2,\dots,x_{m-1}) &\geq 0, \quad i=1,2,\dots,n-1\\ -f(x_1,x_2,\dots,x_{m-1})-x_m &\geq 0, \end{aligned} \tag{2.2}$$

where $(x_1,\dots,x_m)\in I_- = D\times[0,-\underline{x}_m] = \big([\underline{x}_1,\overline{x}_1],\dots,[\underline{x}_{m-1},\overline{x}_{m-1}],[0,-\underline{x}_m]\big)$. The integral value is the difference of the first and second volumes. Since problems (2.1) and (2.2) are very similar, it is sufficient to make an algorithm for the first problem. If $f(x_1,x_2,\dots,x_{m-1})\geq 0$ on D, then $\underline{x}_m = 0$ and only the system of inequalities (2.1) is used indeed. Our algorithm is made for the case $\underline{x}_m = 0$ and the codes of the algorithm run on the adequate segment once (with the data of (2.1)) or twice (with the data of (2.1) and (2.2)). The notation

$$f_n(x_1,x_2,\dots,x_{m-1},x_m) := f(x_1,x_2,\dots,x_{m-1})-x_m$$

is also used from now on. The volume of the box $I = I_+$ is known (denoted by $\operatorname{vol}(I)$), furthermore boxes to the solution set S of (2.1) and boxes to the complementary set $\overline{S} = I\setminus S$ can be computed by using zone functions. If the approximating value to the integral value is denoted by $\mu(S)$ and its error bound by ε, then $\mu(S)=0$, $\varepsilon=\operatorname{vol}(I)$ are the initial values. If the computed box Z fulfils $Z\subset S$, then $\mu(S)=\mu(S)+\operatorname{vol}(Z)$, $\varepsilon=\varepsilon-\operatorname{vol}(Z)$ and if $Z\subset\overline{S}$, then $\varepsilon=\varepsilon-\operatorname{vol}(Z)$. Hence the error bound is improved in every step and the approximating value of the integral is improved in cases of $Z\subset S$. A detailed description of the algorithm is as follows.

(a) Define the first element of a box (interval) sequence $\{I_k\}$ by $I_1 = I$. Let $n_b=1$, $e_b=0$, $\mu(S)=0$, $\varepsilon=\operatorname{vol}(I)$, where n_b, e_b, $\mu(S)$, ε denote the number of boxes in the sequence, the number of the examined boxes, the approximating value of $\operatorname{vol}(S)$, and the error bound, respectively.

(b) Let $e_b = e_b+1$. Compute $\min f_i(c)$, where $1\leq i\leq n$ and c is the centre of I_{n_b}.
   (1) If $\min f_i(c)<0$, then compute the box (zone) $Z = Z_{f_i^*}(c,0)\subset\overline{S}$, where the function f_i^* is the first among $f_1, f_2,\dots,f_n$, having the smallest function value at c. (The "worst inequality" is used here.) Let $\varepsilon=\varepsilon-\operatorname{vol}(Z)$.
   (2) If $\min f_i(c)\geq 0$, then $Z := I_{n_b}$ and $Z := Z\cap Z_{f_i}(c,0)$, $i=1,2,\dots,n$. Let $\mu(S)=\mu(S)+\operatorname{vol}(Z)$, $\varepsilon=\varepsilon-\operatorname{vol}(Z)$.

(c) Divide the set $I_{n_b}\setminus Z$ in L boxes (if the set is empty, then $L:=0$). Filter the "unimportant" (too small) boxes by the simple condition $\operatorname{vol}(\text{box})>\kappa$, where κ is a (small) given value. Place the $L^*\leq L$ new boxes into the box sequence as n_bth, (n_b+1)th, $\dots$, (n_b+L^*-1)th elements and let $n_b = n_b+L^*-1$. If $n_b=0$, then give $\mu(S)$, ε, e_b and stop, otherwise go to (b).

The function and volume values used several times are computed once and are stored. The partitioning of the set $I_{n_b} \setminus Z$ comes from a special case of the box complementation algorithm (see [4, 6]). The box sequence $\{I_k\}$ always contains only the boxes which are waiting for examination (the new boxes are indexed from n_b). There are two essential differences between this algorithm and the algorithm of [4]:

(i) Before step (b), the (first) maximum volume element of the box sequence $\{I_k\}$ is selected (the "most promising box" is used at the beginning of every new loop) in [4].

(ii) The too small boxes are filtered by the simple condition $\mathrm{vol}\,(\mathrm{box}) > \kappa$ here, and by the sophisticated condition $\mathrm{vol}\,(\mathrm{box}) * (E_b - e_b) > \varepsilon$, where E_b is a given upper bound to e_b, in [4].

Now survey the general case ($\underline{x}_m < 0$) and the final results more exactly. If (temporarily) $\mu(S)^+$, ε^+, e_b^+ and $\mu(S)^-$, ε^-, e_b^- denote the output values of the first and second running of the adequate segment, respectively, then $\mu(S)^+ - (\mu(S)^- + \varepsilon^-)$ is lower bound and $(\mu(S)^+ + \varepsilon^+) - \mu(S)^-$ is upper bound for the exact integral value (because $\mu(S)^+$, $\mu(S)^-$ are lower approximations). Hence the mean value $\mu(S) = \mu(S)^+ + \varepsilon^+/2 - (\mu(S)^- + \varepsilon^-/2)$ is an improved value for the integral, the mean value $\varepsilon = \pm(\varepsilon^+/2 + \varepsilon^-/2)$ is an improved value for the error and the value $e_b = e_b^+ + e_b^-$ is the full number of the boxes examined, i. e., these $\mu(S)$, ε, e_b are printed as final results (if $\underline{x}_m = 0$, then $\mu(S)^- = \varepsilon^- = e_b^- = 0$). The improved $\mu(S)$ is obtained by an "error equalization", therefore it often has much less error than ε.* Our numerical examples illustrate very convincingly that simplicity is useful in this integral algorithm, i. e., the selection mentioned in [4] can be too expensive. Table 1 contains the essential data for the following three integrals (where $I_\pm = I_+ \cup I_-$):

$$\int_V \arctan\left(\cos x - 3^x\right) dx$$

with $I_\pm = [[-1, 3], [-10, 10]]$;

$$\iint_V \frac{2 + \cos(x-3)\cos(y+2)}{1 + |x| + 4|y|} dxdy, \qquad \begin{cases} 16 - x^2 - 4y^2 \geq 0, \\ x^2 - y^2 - 4 \geq 0, \end{cases}$$

with $I_\pm = [[-5, 5], [-5, 5], [0, 3]]$;

$$\iiint_V |x| \sqrt{y^2 + |z|}\, dxdydz, \qquad 4 - 2x^2 - 3y^2 - 4z^2 \geq 0,$$

with $I_\pm = [[-2, 2], [0, 2], [-2, 0], [0, 6]]$.

The programs worked on a PC Pentium 4 with a 3.2 GHz processor and running times belong to our Visual C++ version 6.0 code.

The experience is as follows. (1) Naturally, other numerical methods (e. g., the quadrature methods for integrals with one variable) could compute approximating

*See the "likely errors" coming from the comparison of different results at the following integrals.

Definite integral	$\mu(S)\pm\varepsilon$, e_b, time, for $\kappa=10^{-5}$	$\mu(S)\pm\varepsilon$, e_b, time, for $\kappa=10^{-6}$	$\mu(S)\pm\varepsilon$, e_b, time, with code of [4]
$\int_V \cdots$	-3.2988±0.0067 2953, 0.02 sec	-3.2987±0.0021 9392, 0.06 sec	-3.2987±0.0021 9813, 0.15 sec
$\iint_V \cdots$	2.4566±0.1706 50052, 0.8 sec	2.4590±0.0778 222029, 4 sec	2.4590±0.0772 228740, 149 sec
$\iiint_V \cdots$	0.6579±0.1329 31486, 0.4 sec	0.6524±0.0703 166939, 2 sec	0.6522±0.0706 169198, 91 sec

TABLE 1. Approximating integral values with error bounds

integral values more quickly, without guaranteed error bounds. The author does not know alternative computer codes for solving the problem stated here.

(2) In the 2nd and 3rd examples $I_\pm = I_+$ and we use rough enough upper bounds for the integrands. In the first example $I_\pm = I_+ \cup I_-$ and we use rough lower and upper bounds (-10 and 10 instead of $-\pi/2$ and $\pi/2$) for the integrand. The experience is that the accuracy essentially does not depend on the circumstances mentioned. For example, if $-\pi/2$ and $\pi/2$ are used in place of -10 and 10, respectively, the result is practically the same.

(3) By the geometrical meaning, the solution set S was covered with rectangles in the 1st example, with cuboids in the 2nd example, with four dimensional (abstract) boxes in the third integral. In general, the computational effort to achieve a good covering is greater and greater as the number of dimensions increases. (The second example requires prominent computational effort because the integrand is "wavy" over a large box of 100 units in volume, and the set V contains two disjunct parts.)

(4) The real strength of our algorithm compared to [4] is illustrated in the 2nd and 3rd columns of Table 1. The results of the second column come by the algorithm of [4] after 2.5, 37.25, 45.5 times more time, respectively. Hence the selection step of [4] could be too expensive indeed.

3. ALGORITHM FOR FINDING GLOBAL MAXIMA

Consider the problem

$$\begin{aligned} &\text{maximize} && f(x_1,x_2,\dots,x_{m-1}) \\ &\text{subject to} && f_i(x_1,x_2,\dots,x_{m-1}) \ge 0, \quad i=1,2,\dots,n-1,\ m\ge 2,\ n\ge 1; \\ & && (x_1,\dots,x_{m-1}) \in D = \left([\underline{x}_1,\overline{x}_1],\dots,[\underline{x}_{m-1},\overline{x}_{m-1}]\right) \subset \mathbb{R}^{m-1}, \end{aligned}$$

where the multivariate real functions f_i, $i=1,\dots,n-1$ and f are continuous on the box D and built from the elementary functions mentioned. Let us assume that we

know (rough) lower and upper bounds $\underline{x}_m$, $\overline{x}_m$ that

$$\underline{x}_m \le f(x_1, x_2, \ldots, x_{m-1}) \le \overline{x}_m, \quad \forall (x_1, x_2, \ldots, x_{m-1}) \in D.$$

Define the system of inequalities

$$f_i(x_1, x_2, \ldots, x_{m-1}) \ge 0, \quad i = 1, 2, \ldots, n-1$$
$$f(x_1, x_2, \ldots, x_{m-1}) - x_m \ge 0,$$

where $(x_1, x_2, \ldots, x_m) \in I = \big([\underline{x}_1, \overline{x}_1], \ldots, [\underline{x}_{m-1}, \overline{x}_{m-1}], [\underline{x}_m, \overline{x}_m]\big)$, or briefly

$$f_i(x_1, x_2, \ldots, x_m) \ge 0, \quad i = 1, 2, \ldots, n, \ (x_1, x_2, \ldots, x_m) \in I, \tag{3.1}$$

where $f_n(x_1, x_2, \ldots, x_m) = f(x_1, x_2, \ldots, x_{m-1}) - x_m$. The solution of our problem is a point of the solution set S of (3.1) with the largest mth coordinate. By a fine scanning of S seen in the integral algorithm, a good approximation value ω can be obtained for the maximum function value (belonging to the set of feasible points) and henceforth it is proved that the value $\omega + \varepsilon$ (where ε is a supposed error bound) is an upper bound to the maximum value. More exactly the aim is to do a fine scanning only around the solution point, therefore a second filter is used besides the simple one seen in the integral algorithm. A detailed description of the algorithm is as follows (it is supposed that the set of feasible points is not empty).

(a) Define the first element of a box (interval) sequence $\{I_k\}$ by $I_1 = I$. Let $n_b = 1$, $e_b = 0$, $\omega = -\infty$, where n_b, e_b, ω denote the number of boxes in the sequence, the number of the examined boxes, and the approximating value to the maximum function value, respectively.

(b) Let $e_b = e_b + 1$. Compute $\min f_i(c)$, where $1 \le i \le n$ and c is the centre of I_{n_b}. If $\min f_i(c) \ge 0$ (c is a feasible point) and $f(c) > \omega$, then $p := c$, $\omega := f(c)$.
  (1) If $\min f_i(c) < 0$, then compute the box (zone) $Z = Z_{f_i^*}(c, 0) \subset \overline{S}$, where the function f_i^* is the first among $f_1, f_2, \ldots, f_n$ having the smallest function value at c. (The "worst inequality" is used here.)
  (2) If $\min f_i(c) \ge 0$, then $Z := I_{n_b}$ and $Z := Z \cap Z_{f_i}(c, 0)$, $i = 1, 2, \ldots, n$.

(c) Divide the set $I_{n_b} \setminus Z$ in L boxes (if the set is empty, then $L := 0$). Filter the 'unimportant' boxes by the conditions $\mathrm{vol}\,(\mathrm{box}) > \kappa$ (where κ is a small given value) and $\overline{x}_m^* > \omega$ (where $\overline{x}_m^*$ is the maximum value of the mth coordinate in the box). Place the $L^* \le L$ new boxes into the box sequence as n_bth, $(n_b + 1)$th, $\ldots$, $(n_b + L^* - 1)$th elements and let $n_b = n_b + L^* - 1$. If $n_b = 0$ or $e_b \ge E_b$ (E_b is an upper bound to the number of the examined boxes), then give p, ω, e_b and go to (d), otherwise go to (b).

(d) Modify I and κ. Let $I = \big([\underline{x}_1, \overline{x}_1], \ldots, [\underline{x}_{m-1}, \overline{x}_{m-1}], [\omega + \varepsilon, \overline{x}_m]\big)$ and $\kappa = 0$. Run (a)-(c) with the new initial values again. If the second return to (d) is realized by $n_b = 0$ (in such a case $\omega = -\infty$), then give $\omega + \varepsilon$ as an upper bound to the maximum function value and e_b as the number of the examined

boxes in the second running. If the second return to (d) is realized by $e_b \geq E_b$, then give e_b. Stop.

The function and volume values used several times are computed once and are stored. The partitioning of the set $I_{n_b} \setminus Z$ comes from a special case of the box complementation algorithm (see [4, 6]). The box sequence $\{I_k\}$ always contains only the boxes which are waiting for examination (the new boxes are indexed from n_b). There are three essential differences between this algorithm and the algorithm of [3]:

(i) The latter one is a strongly heuristic (with hindsight, also a little confused) method for our problem without using solution boxes of nonlinear systems of inequalities.
(ii) A selection step is used at the beginning of every new loop in [3].
(iii) The algorithm of [3] does not prove upper bound to the maximum function value (this is why there are no comparisons between the two algorithms in the next table).

Global optimum problem	$\omega + \varepsilon$, $e_b + e_b$, time, for $\kappa = 10^{-5}$	$\omega + \varepsilon$, $e_b + e_b$, time, for $\kappa = 10^{-6}$
$\arctan(\cos x - 3^x)$ ...	0.2988+0.01 265+64, 0.002sec	0.2988+0.01 542+64, 0.004sec
$\dfrac{2+\cos(x-3)\cos(y+2)}{1+\lvert x\rvert+4\lvert y\rvert}$ ...	0.6206+0.01 3160+497, 0.05sec	0.6256+0.01 6481+419, 0.11sec
$\lvert x\rvert\sqrt{y^2+\lvert z\rvert}$ ...	0.9222+0.01 3104+578715, 6sec	0.9312+0.01 7643+45165, 0.5sec

TABLE 2. Approximations of global maxima with error bounds

Table 2 contains the essential data for the three problems (the functions of the integral problems are used again):

$$\arctan\left(\cos x - 3^x\right) \to \max$$

with $I = [[-1,3],[-10,10]]$;

$$\frac{2+\cos(x-3)\cos(y+2)}{1+|x|+4|y|} \to \max, \qquad \begin{cases} 16-x^2-4y^2 \geq 0, \\ x^2-y^2-4 \geq 0, \end{cases}$$

with $I = [[-5,5],[-5,5],[0,3]]$;

$$|x|\sqrt{y^2+|z|} \to \max, \qquad 4-2x^2-3y^2-4z^2 \geq 0,$$

with $I = [[-2,2],[0,2],[-2,0],[0,6]]$.

The programs worked on a PC Pentium 4 with a 3.2 GHz processor and running times belong to our Visual C++ version 6.0 code.

The experience is as follows.

(1) Comparing the first member of $e_b + e_b$ to the value e_b seen in the integral problem (265, 3160, 3104 instead of 2953, 50052, 31486 for $\kappa = 10^{-5}$, respectively) the effect of the second filter $\overline{x}_m^* > \omega$ is illustrated. Without this filter the examination in the first step would be similar to the examination of the integral algorithm, the filter $\overline{x}_m^* > \omega$ can utilize the intermediate ω values very well. (A good second filter to integral algorithms could be applied only with difficulty.)

(2) A very sharp or faulty value $\omega + \varepsilon$ as upper bound to the maximum function value belonging to the set of feasible points could cause huge computational efforts. The stop criterion $e_b \geq E_b$ can prevent the needless work. In the above 6 cases the value $E_b = 10^6$ was used, and it allows the entire computation with the sharp upper bound of the 5th problem ($e_b = 578715$ shows the computational effort in proving the solution set is empty in a system of inequalities which very nearly has some solutions).

(3) A faulty result never happened to be obtained because of the effect of rounding errors. The boxes are computed by lower estimates, see the proofs of rules in [2], and our C++ program uses double type for real variables. (The circumstances are the same for the integral algorithm.)

4. Codes for the two algorithms

Our Visual C++ version 6.0 and Lahey Fortran 90 version 4.5 programs have 5 segments for both algorithms. The first 3 segments are the same in the integral and global maximum algorithms. The function (subroutine) segment fval computes the function values from the numerically coded form, i. e., it handles the function

$$\text{fval} : (G, c) \mapsto g(c),$$

where G is a numerically coded form of the multivariate real function g, and c is a point of the domain. For computing the zone function values (boxes) the function (subroutine) segment zone is used, which handles the function

$$\text{zone} : (D, G, c, \alpha) \mapsto Z_g(c, \alpha),$$

where D is the domain box of the multivariate real function g, G is a numerically coded form of g, $c \in D$ and $\alpha \in \mathbb{R}$. To divide the difference of a closed m-dimensional box U and an open m-dimensional box T into closed boxes $B_1, B_2, \ldots, B_L$ which do not contain common interior points, our function (subroutine) segment ints handles the function

$$\text{ints} : (U, T) \mapsto (L, \{B_1, B_2, \ldots, B_L\}).$$

In the integral algorithm, the function (subroutine) segment appr, for computing approximating values to problem (2.1), handles the function

$$\text{appr} : (F, I, \kappa) \mapsto (\mu(S), \varepsilon, e_b),$$

where F is the numerically coded form of

$$(x_1,\dots,x_m)\mapsto(f_1(x_1,\dots,x_m),\dots,f_n(x_1,\dots,x_m))$$

and I, κ, $\mu(S)$, ε, e_b come from our algorithm description. The main (program) segment produces data for calling the segment appr. These data are m, n, F, $I_\pm = I_+ \cup I_-$ and κ. The segment appr is called only once if $I_\pm = I_+$ and it is called twice if $\mathrm{vol}\,(I_-) \neq 0$. At the global maximum algorithm, the function (subroutine) segment glob, for computing approximating values to problem (3.1), handles the function

$$\mathrm{glob} : (F, I, \kappa, E_b) \mapsto (p, \omega, e_b)\,,$$

where F is the numerically coded form of

$$(x_1,\dots,x_m)\mapsto(f_1(x_1,\dots,x_m),\dots,f_n(x_1,\dots,x_m))$$

and I, κ, E_b, p, ω, e_b come from our algorithm description. The main (program) segment produces data for calling the segment glob. These data are m, n, F, I, κ and E_b. First the segment glob is called for finding p and ω, and the aim of the second call is to prove that the value $\omega + \varepsilon$ is upper bound to the exact solution. Naturally, these segments have some other (auxiliary) parameters and sometimes their notations cannot follow names in this paper perfectly. Finally, some remarks and comparisons between C++ and Fortran programs are given.

(1) The organization of the C++ and Fortran programs differs from one another essentially in two cases only. The Fortran subroutine segments can handle all output parameters and large arrays in natural mode (as local variables), the C++ function segments use global variables for these data. The lengths of numerical coded forms of multivariate functions are stored as zero indexed elements of the arrays in question in C++ programs and they are stored in an independent array in Fortran programs. Naturally, the first 4 segments of both C++ and Fortran programs are unaltered in the 3 different problems, only the calling segment has to be changed.

(2) Our C++ programs use double type for real variables, and double precision variables are in the Fortran programs. The results of Table 1 and Table 2 (obtained by C++ programs) are the same essentially as those by Fortran programs. The Fortran codes are hardly faster than the C++ ones (only 0-10 percent are the differences in running time). This fact is somewhat surprising, because in the former methods (with selections) or in the zone function tests (with simple real variables) differences of 30-50 percent often appeared.

(3) The author would gladly send the Maple, C++, and Fortran codes mentioned to interested readers in e-mail as an attached file.

REFERENCES

[1] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *Numerical toolbox for verified computing. I*, ser. Springer Series in Computational Mathematics. Berlin: Springer-Verlag, 1993, vol. 21, basic numerical problems, Theory, algorithms, and Pascal-XSC programs, With separately available computer disks.

[2] F. Kálovics, "Creating and handling box valued functions used in numerical methods," *J. Comput. Appl. Math.*, vol. 147, no. 2, pp. 333–348, 2002.
[3] F. Kálovics, "Solving nonlinear constrained minimization problems with a new interval valued function," *Reliab. Comput.*, vol. 5, no. 4, pp. 395–406, 1999.
[4] F. Kálovics, "Zones and integrals," *J. Comput. Appl. Math.*, vol. 182, no. 2, pp. 243–251, 2005.
[5] F. Kálovics and G. Mészáros, "Box valued functions in solving systems of equations and inequalities," *Numer. Algorithms*, vol. 36, no. 1, pp. 1–12, 2004.
[6] R. B. Kearfott, *Rigorous global search: continuous problems*, ser. Nonconvex Optimization and its Applications. Dordrecht: Kluwer Academic Publishers, 1996, vol. 13.

Author's address

F. Kálovics
University of Miskolc, Department of Analysis, H-3515 Miskolc–Egyetemváros, Hungary
E-mail address: matkf@uni-miskolc.hu