An always convergent algorithm for global minimization of univariate Lipschitz functions

József Abaffy
Institute of Applied Mathematics, Óbuda University, H-1034 Budapest, Bécsi út 96/b, Hungary
abaffy.jozsef@nik.uni-obuda.hu

Aurél Galántai
Institute of Applied Mathematics, Óbuda University, H-1034 Budapest, Bécsi út 96/b, Hungary
galantai.aurel@nik.uni-obuda.hu

Abstract: We develop and analyze a bisection type global optimization algorithm for real Lipschitz functions. The suggested method combines the branch and bound method with an always convergent solver of nonlinear equations. The computer implementation and performance are investigated in detail.

Keywords: global optimum; nonlinear equation; always convergent method; Newton method; branch and bound algorithms; Lipschitz functions

1 Introduction

In paper [2] we defined the following branch and bound method to find the global minimum of the problem

\[ f(z) \rightarrow \min \]
\[ l \leq z \leq u, \]

where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is sufficiently smooth and \( l, u \in \mathbb{R}^n \). Assume that

\[ z_{\text{output}} = \text{alg.min}(f, z_{\text{input}}) \]

is a local minimization algorithm that satisfies \( f(z_{\text{output}}) \leq f(z_{\text{input}}) \), for any \( z_{\text{input}} \). Similarly, assume that

\[ [z_{\text{sol}}, I_{\text{flag}}] = \text{equation.solve}(f, c) \]
denotes a solution algorithm of the single multivariate equation \( f(z) = c \) such that \( iflag = 1 \), if a true solution \( z_{sol} \) exists (that is \( f(z_{sol}) = c \)), and \( iflag = -1 \), otherwise.

Let \( f_{\text{min}} \) denote the global minimum of \( f \), and let \( B_{\text{lower}} \in \mathbb{R} \) is a lower bound of \( f \) such that \( f_{\text{min}} \geq B_{\text{lower}} \). Let \( z_0 \in D_f \) be any initial approximation to the global minimum point (\( f(z_0) \geq B_{\text{lower}} \)). The suggested algorithm of [2] then takes the form:

Algorithm 1

\[
\begin{align*}
    z_1 &= \text{alg}_\text{min}(f, z_0) \\
    a_1 &= f(z_1), \quad b_1 = B_{\text{lower}}, \quad i = 1 \\
    \text{while } a_i - b_i > \text{tol} & \quad \Rightarrow \\
    c_i &= (a_i + b_i) / 2 \\
    [\xi, iflag] &= \text{equation}_\text{solve}(f, c_i) \\
    \text{if } iflag = 1 & \quad \Rightarrow \\
    z_{i+1} &= \text{alg}_\text{min}(f, \xi), \quad a_{i+1} = f(z_{i+1}), \quad b_{i+1} = b_i \\
    \text{else} & \quad \Rightarrow \\
    z_{i+1} &= z_i, \quad a_{i+1} = a_i, \quad b_{i+1} = c_i \\
    \text{end} \\
    i &= i + 1 \\
\end{align*}
\]

Using the idea of Algorithm 1 we can also determine a lower bound of \( f \), if such a bound is not known a priori (for details, see [2]). Algorithm 1 shows conceptual similarities with other multidimensional bisection type algorithms such as those of Shary [34] and Wood [50], [52].

**Theorem 1.** Assume that \( f : \mathbb{R}^n \to \mathbb{R} \) is continuous and bounded from below by \( B_{\text{low}} \). Then Algorithm 1 is globally convergent in the sense that \( f(z_i) \to f_{\text{min}} \).

**Proof.** At the start we have \( z_1 \) and the lower bound \( b_1 \) such that \( f(z_1) \geq f_{\text{min}} \geq b_1 \). Then we take the midpoint of this interval, i.e. \( c_1 = (f(z_1) + b_1) / 2 \). If a solution \( \xi \) exists such that \( f(\xi) = c_1 \) (\( iflag = 1 \)), then \( c_1 \geq f_{\text{min}} \) holds. For the output \( z_2 \) of the local minimizer, the inequality \( c_1 \geq f(z_2) \geq f_{\text{min}} \geq b_1 \) holds by the initial assumptions. If there is no solution of \( f(\xi) = c_1 \) (i.e. \( iflag = -1 \)), then \( c_1 < f_{\text{min}} \). By continuing this way we always halve the inclusion interval \((b_i, f(z_i))\) at the worst case. So the method is convergent in the sense that \( f(z_i) \to f_{\text{min}} \). Note that sequence \( \{z_i\} \) is not necessarily convergent.
The practical implementation of Algorithm 1 clearly depends on the local minimizer, the equation solver and also on $f$. Since we have several local minimizers satisfying the above requirements we must concentrate on the equation solvers. There are essentially two questions to be dealt with. Namely, the existence of the solution and the very existence of methods that are always convergent in the sense that either they give a solution when exists or give a warning sign if no solution exists.

The existence of solution follows from the Weierstrass theorem, if $f_{\text{min}} \leq c \leq f(z_0)$. As for the solvers we may observe that for $n > 1$, our equation is an underdetermined nonlinear equation of the form

$$g(z) = f(z) - c = 0 \quad (g : \mathbb{R}^n \to \mathbb{R}).$$

There are several locally convergent methods for such equations (see, e.g. [25], [3], [45], [26], [27], [28], [47], [48], [12], [13], [14]). In paper [2] we tested Algorithm 1 with a nonlinear Kaczmarz projection algorithm [45], [26], [27], [25], which showed fast convergence in most of the test cases, but also showed numerical instability in some cases, when $\nabla f(z_k)$ was close to zero.

There also exist always convergent methods for equation (1) (see, e.g. [37], [9], [20], [22], [21], [43], [44], [1], [31], [46]). For the multivariate case, most methods are related to subdivision and seem to be quite slow. For univariate equations, however, the always convergent methods of Szabó [43], [44], Abaffy and Forgó [1], Pietrus [31] and Várterész [46] are using other principles than subdivision and they are quite fast.

Here we study Algorithm 1 for one-dimensional real Lipschitz functions. The global minimization of real Lipschitz functions has a rich literature with many interesting and useful algorithms. For these, we refer to Hansen, Jaumard, Lu [15], [17], [18] and Pintér [32].

The outline of paper is the following. We develop and analyze the equation solver in Section 2. In Section 3 we develop a modified implementation of Algorithm 1 called Algorithm 2 that use this equation solver and double bisection. The final section contains the principles and results of numerical testing. The comparative numerical testing indicates that Algorithm 2 can be a very efficient minimizer in practice.

### 2 An always convergent solver for real equations

Consider the real equation

$$g(t) = 0 \quad (g : \mathbb{R} \to \mathbb{R}, \quad t \in [\alpha, \beta])$$

An iterative solution method of the form $x_{n+1} = F(g; x_n)$ is said to be always convergent, if for any $x_0 \in [\alpha, \beta]$ ($g(x_0) \neq 0$)

- (i) the sequence $\{x_n\}$ is monotone,
(ii) \( \{x_n\} \) converges to the zero in \([\alpha, \beta]\) that is nearest to \(x_0\), if such zero exists,

(iii) if no such zero exists, then \( \{x_n\} \) exits the interval \([\alpha, \beta]\).

Assuming high order differentiability, Szabó [43], [44] and Várterész [46] developed some high order always convergent iterative methods. Assuming only continuous differentiability Abaffy and Forgó [1] developed a linearly convergent method, which was generalized to Lipschitz functions by Pietrus [31] using generalized gradient in the sense of Clarke.

Since we assume only the Lipschitz continuity of \(g\), we select and analyze an always convergent modification of the Newton method. This method was first investigated by Szabó [43], [44]) under the condition that \(g\) is differentiable and bounded in the interval \([\alpha, \beta]\). We only assume that \(g\) satisfies the Lipschitz condition.

**Theorem 2.** (a) Assume that \( |g(t) - g(s)| \leq M |t - s| \) holds for all \( t, s \in [\alpha, \beta] \). If \( x_0 \in (\alpha, \beta) \) and \( g(x_0) \neq 0 \), then the iteration

\[
x_{n+1} = x_n - \frac{|g(x_n)|}{M} \quad (n = 0, 1, \ldots)
\]

either converges to the zero of \(g\) that is nearest to \(x_0\) or the sequence \( \{x_n\} \) exits the interval \([\alpha, \beta]\). (b) If \( y_0 \in [\alpha, \beta] \) and \( g(y_0) \neq 0 \), then the iteration

\[
y_{n+1} = y_n + \frac{|g(y_n)|}{M} \quad (n = 0, 1, \ldots)
\]

either converges to the zero of \(g\) that is nearest right to \(y_0\) or the sequence \( \{y_n\} \) exits the interval \([\alpha, \beta]\).

**Proof.** We prove only part (a). The proof of part (b) is similar. It is clear that \( x_{n+1} \leq x_n \). If a number \( \gamma \) exists such that \( \alpha \leq \gamma \leq x_0 \) and \( x_n \to \gamma \), then \( g(\gamma) = 0 \). Otherwise there exists an index \( j \) such that \( x_j < \alpha \). Assume now that \( \alpha \leq \gamma < x_0 \) is the nearest zero of \(g\) to \(x_0\). Also assume that \( \gamma \leq x_n \) \((n \geq 1) \). We can write

\[
x_{n+1} - \gamma = x_n - \gamma - \frac{|g(x_n) - g(\gamma)|}{M} = \left(1 - \frac{\xi_n}{M}\right) (x_n - \gamma) \quad (\xi_n \in [0, M]).
\]

Since \( 0 \leq 1 - \frac{\xi_n}{M} \leq 1 \), we obtain that \( \gamma \leq x_{n+1} \) and \( x_{n+1} - \gamma \leq x_n - \gamma \). Hence the method, if converges, then converges to the nearest zero to \(x_0\). Assume that no zero exists in the interval \([\alpha, x_0]\) and let \( |g|_{\min} = \min_{\alpha \leq t \leq x_0} |g(t)| \). Then

\[
x_{n+1} = x_n - \frac{|g(x_n)|}{M} \leq x_n - \frac{|g|_{\min}}{M} \leq x_0 - (n + 1) \frac{|g|_{\min}}{M},
\]

and algorithm (3) leaves the interval in at most \( \frac{M(x_0 - \alpha)}{|g|_{\min}} \) steps. A similar claim holds for algorithm (4). \qed
The convergence speed is linear in a sense. Assume that $\alpha \leq \gamma < x_0$ is the nearest zero to $x_0$ and $\epsilon > 0$ is the requested precision of the approximate zero. Also assume that a number $m_\epsilon > 0$ exists such that $m_\epsilon \gamma \leq |g(t)| \leq M |t - \gamma|$ holds for all $\gamma + \epsilon \leq t \leq x_0$. If $g$ is continuously differentiable in $[\alpha, \beta]$, then $m_\epsilon = \min_{t \in [\gamma+\epsilon, x_0]} \|g'(t)\|$. Having such a number $m_\epsilon$ we can write (5) in the form

$$x_n - \gamma \leq \left(1 - \frac{m_\epsilon}{M}\right)^n (x_0 - \gamma) \leq \left(1 - \frac{m_\epsilon}{M}\right)^n (\beta - \alpha).$$

This indicates a linear speed achieved in at most

$$\left[\frac{\log \frac{\epsilon}{\beta - \alpha}}{\log(1 - \frac{m_\epsilon}{M})}\right]$$

steps. We can assume that $m_\epsilon > \epsilon$, which gives the bound

$$\left[\frac{\log \frac{\epsilon}{\beta - \alpha}}{\log(1 - \frac{m_\epsilon}{M})}\right].$$

Relation $\log (1 + \epsilon) \approx \epsilon$ yields the approximate expression $M \left[\log \frac{\epsilon}{\beta - \alpha}\right] \epsilon^{-1}$ for the number of required iterations.

For the optimum step number of algorithms in the class of Lipschitz functions, see Sukharev [42] and Sikorski [35].

Assume now that $L > 0$ is the smallest Lipschitz constant of $g$ on $[\alpha, \beta]$ and $M = L + c$ with a positive $c$. It then follows from (5) that

$$x_{n+1} - \gamma \geq \left(1 - \frac{L}{L+c}\right) (x_n - \gamma) = \left(\frac{c}{L+c}\right)^{n+1} (x_0 - \gamma).$$

This indicates a linear decrease of the approximation error. Note that the method can be very slow, if $c/(L+c)$ is close to 1 (if $M$ significantly overestimates $L$) and it can be fast, if $c/(L+c)$ is close to 0 (if $M$ is close to $L$). Equation (5) also shows that $M$ can be replaced in the algorithms (3)-(4) by the appropriate $M_n$ that satisfies the condition $0 \leq \frac{c}{M_n} \leq 1$. For differentiable $g$, $M_n$ might be close to $|g'(x_0)|$ in order to increase the speed (case of small $c$).

A simple geometric interpretation shows that the two algorithms are essentially the same. The Lipschitz condition implies that $||g(t)| - |g(s)|| \leq M |t - s|$ ($t, s \in [\alpha, \beta]$) also holds. The resulting inequality

$$|g(x)| - M |x - t| \leq |g(t)| \leq |g(x)| + M |x - t|$$

gives two linear bounding functions for $|g(t)|$, namely $|g(x)| + M(x-t)$ and $|g(x)| + M(t-x)$ for a fixed $x$. If the zero $\gamma$ is less than $x_n$, then for $t \leq x_n$, the linear function $|g(x_n)| + M(t-x_n)$ will be under $|g(t)|$. Its zero $x_{n+1} = x_n - \frac{|g(x_n)|}{M} \leq x_n$ is the next approximation to $\gamma$ and $x_{n+1} \geq \gamma$ clearly holds. Similarly, if $y_n < \gamma$, then $|g(y_n)| + M(y_n-t)$ will be under $|g(t)|$ and for its zero, $y_n \leq y_{n+1} = y_n + \frac{|g(y_n)|}{M} \leq \gamma$ clearly holds. The next figure shows both situations with respect to an enclosed unique zero $\gamma$. 
It also follows that if $g(x_0) > 0$ ($g(x_0) < 0$) then $g(t) > 0$ ($g(t) < 0$) for $\gamma < t \leq x_0$, if such a zero $\gamma$ exists. If not, $g(t)$ keeps the sign of $g(x_0)$ in the whole interval $[\alpha, x_0]$. An analogue result holds for algorithm (4).

Consider the following general situation with arbitrary points $u, v \in [\alpha, \beta]$ ($u < v$).

The points $(u, g(u))$ and $(v, g(v))$ and the related linear bounding functions define a parallelogram that contains function $g$ over the interval $[u, v]$ with the bounds

$$
\frac{g(u) + g(v)}{2} + M \frac{u - v}{2} \leq g(t) \leq \frac{g(u) + g(v)}{2} + M \frac{v - u}{2} \quad (u \leq t \leq v).
$$
This property is the basis of Piyavskii’s minimization algorithm and related methods (see, e.g. [17], [32]). It is also exploited in Sukharev’s modified bisection method [41], [42].

Function \( g(t) \) may have a zero in \([u, v]\) only if

\[
g(u) + g(v) + M(u - v) \leq 0 \leq g(u) + g(v) + M(v - u),
\]

that is if

\[
|g(u) + g(v)| \leq M(v - u). \tag{6}
\]

If \( g(t) \) has a zero \( \gamma \in (u, v) \), then by the proof of Theorem 2.

\[
u + \frac{|g(u)|}{M} \leq \gamma \leq v - \frac{|g(v)|}{M} \tag{7}
\]

holds and (6) is clearly satisfied. If \( u \) and \( v \) are close enough and \((u, v)\) does not contain a zero of \( g(t) \), then (6) does not hold. This happens, if \( u \geq v - \frac{|g(v)|}{M} \) and \( g(u) \neq 0 \) or \( v \leq u + \frac{|g(u)|}{M} \) and \( g(v) \neq 0 \).

Note that iterations (3)-(4) satisfy the bounds

\[
g\left(\frac{x_{n+1} + x_n}{2} \right) + \frac{|g(x_n)|}{M} \leq g(t) \leq g\left(\frac{x_{n+1} + x_{n+1}}{2} \right) + \frac{|g(x_n)|}{M} \tag{8}
\]

for \( x_{n+1} \leq t \leq x_n \), and the bounds

\[
g\left(\frac{y_n + y_{n-1}}{2} \right) + \frac{|g(y_n)|}{M} \leq g(t) \leq g\left(\frac{y_n + y_{n-1} + 1}{2} \right) + \frac{|g(y_n)|}{M} \tag{9}
\]

for \( y_n \leq t \leq y_{n+1} \).

Note also that if \( u \) and \( v \) are distant enough (in a relative sense), then condition (6) may hold without having a zero in \((u, v)\).

Using the above geometric characterization we can develop practical exit conditions for the nonlinear solver (3)-(4). The most widely used exit conditions are

\[
|x_{n+1} - x_n| < \varepsilon \quad \text{and} \quad |g(x_n)| < \varepsilon,
\]

which are not fail safe neither individually nor in the combined form \( \max \{|x_{n+1} - x_n|, |g(x_n)|\} < \varepsilon \). For a thorough analysis of the matter, see Delahaye [8], Sikorski and Wozniakowski [36] and Sikorski [35]. Another problem arises in the floating precision arithmetic that requires stopping, if either \( |x_{n+1} - x_n| < \varepsilon_{\text{machine}} \) or \( |g(x_n)| < \varepsilon_{\text{machine}} \) holds. Since \( |x_{n+1} - x_n| = \frac{|g(x_n)|}{M} \), the tolerance precision \( \varepsilon \) is viable, if \( \max \{1, M\} \varepsilon_{\text{machine}} \leq \varepsilon \). By the same argument the \( tol \) parameter of Algorithm 1 must satisfy the lower bound \( tol \geq 2\varepsilon_{\text{machine}} \).

If \( g(t) \) has a zero \( \gamma \in [\alpha, x_0] \), the monotone convergence of \( \{x_n\} \) implies the relation \( |x_{n+1} - x_n| \leq |x_n - \gamma| \). Hence \( |x_{n+1} - x_n| \) is a lower estimate of the approximation error.

There are some possibilities to increase the reliability of the combined exit condition. The first one uses algorithm (4) in the following form. If interval \((x_n - \varepsilon, x_n)\)
is suspect to have a zero of \( g(t) \) (and \( g(x_n - \varepsilon), g(x_n) \neq 0 \)), then we can apply condition (6) with \( u = x_n - \varepsilon \) and \( v = x_n \) in the form

\[
M \varepsilon \geq |g(x_n - \varepsilon) + g(x_n)|. \tag{10}
\]

If \( M \varepsilon < |g(x_n - \varepsilon) + g(x_n)| \), then there is no zero in \([x_n - \varepsilon, x_n]\) and we have to continue the iterations. Even if \( M \varepsilon \geq |g(x_n - \varepsilon) + g(x_n)| \) holds, it is not a guarantee for the existence of a zero in the interval \([x_n - \varepsilon, x_n]\).

In the latter case we can apply algorithm (4) with \( y_0 = x_n - \varepsilon \). If there really exists a zero \( \gamma \in (x_n - \varepsilon, x_n) \), then the sequence \( \{y_n\} \) converges to \( \gamma \) and remains less than \( x_n \). If no zero exists in the interval, then \( m = \min_{t \in [x_n - \varepsilon, x_n]} |g(t)| > 0 \) and the iterations \( \{y_n\} \) satisfy \( y_n \geq y_0 + n \frac{m}{M^2} \). Hence the sequence \( \{y_n\} \) exceeds \( x_n \) in a finite number of steps. The same happens at the point \( x_n - \varepsilon \), if we just continue the iterations \( \{x_n\} \).

The two sequences \( \{y_n\} \) and \( \{x_n\} \) exhibit a two-sided approximation to the zero (if exists) and \( x_j - y_k \) is an upper estimate for the error. This error control procedure is fail safe, but it may be expensive. We can make it cheaper by fixing the maximum number of extra iterations at the price of losing absolute certainty. For example, if we use the first extra iteration \( x_{n+1} \) \( (x_n - \varepsilon < x_{n+1}) \) and set \( v = x_{n+1} \), then condition (6) changes to

\[
M \varepsilon \geq |g(x_n - \varepsilon) + g(x_{n+1})| + |g(x_n)|. \tag{11}
\]

Similar expressions can be easily developed for higher number of iterations as well.

A second possibility for improving the exit conditions arises if a number \( m > 0 \) exists such that \( m |t - \gamma| \leq |g(t)| \leq M |t - \gamma| \) holds for all \( t \in [\alpha, \beta] \). Then \( |x_n - \gamma| \leq \frac{1}{m} |g(x_n)| \) is an upper bound for the error. Similarly, we have

\[
|x_n - \gamma| \leq \delta + \frac{1}{m} |g(x_n - \delta)|
\]

and by selecting \( \delta = x_n - x_{n+1} \) we arrive at the bound

\[
|x_n - \gamma| \leq x_n - x_{n+1} + \frac{1}{m} |g(x_{n+1})|.
\]

This type of a posteriori estimate depends however on the existence and value of \( m \).

### 3 The one-dimensional optimization algorithm

We now use algorithms (3)-(4) to implement an Algorithm 1 type method for the one-dimensional global extremum problem

\[
f(t) \rightarrow \min \quad (l \leq t \leq u, \ f: \mathbb{R} \rightarrow \mathbb{R}, \ l, u \in \mathbb{R}) \tag{12}
\]
under the assumption that $|f(t) - f(s)| \leq L |t - s|$ holds for all $t, s \in [l, u]$. Here the solution of equation $f(t) = c$ is sought on the interval $[l, u]$.

It seems handy to apply Algorithm 1 directly with solver (3) or (4). It may happen that equation $f(t) = c_i$ has no solution for some $i$, and this situation is repeated ad infinitum. Since for $\min f > c_i$, the number of iterations is $O\left(\frac{1}{\min f - c_i}\right)$, this may cause severe problems for $c_i > \min f$. Assume that $a_k = a_{k+1} > \min f > c_{k+1} > b_{k+1}$ for $\ell \geq 0$. Then $a_{k+1} - b_{k+1} = a_k - b_{k+1} = \frac{a_k - b_k}{2^{\ell}} \to 0$, which is contradiction to $a_k > \min f > b_{k+1}$ ($\ell \geq 0$). Hence the situation can occur infinitely many times, if by chance $a_k = f(z_k) = \min f$. However preliminary numerical testing indicated a very significant increase of computational time in cases, when $c_i$ just approached $\min f$ from below with a small enough error. This unexpected phenomenon is due to the always convergent property of solver, that we want to keep. Since the iteration numbers also depend on the length of computational interval (see the proof of Theorem 2) we modify Algorithm 1 so that in case $c_i < \min f$ and $c_i \approx \min f$ the computational interval should decrease.

The basic element of the modified algorithm is the solution of equation $g(x) = f(x) - c = 0$ on any subinterval $[\alpha, \beta] \subset [l, u]$. Assume that the upper and lower bounds $a = f(x_\alpha) \geq \min_{x \in [\alpha, \beta]} f(x) > b$ ($x_\alpha \in [\alpha, \beta]$)

are given and $c \in (a, b)$. If equation $f(x) = c$ has a solution in $[\alpha, \beta]$, then

$$\min_{x \in [\alpha, \beta]} f(x) \leq c < a,$$

otherwise

$$\min_{x \in [\alpha, \beta]} f(x) > c > b.$$

If $f(\beta) \neq c$, then we compute iterations $\xi_0 = \beta$ and

$$\xi_{i+1} = \xi_i - \frac{|f(\xi_i) - c|}{M} \quad (i \geq 0). \quad (13)$$

There are two cases:

(i) There exists $x^* \in [\alpha, \beta]$ such that $f(x^*) = c$.

(ii) There exists a number $k$ such that $\xi_k = \alpha$ or $\xi_k < \alpha < \xi_{k-1}$.

In case (i) the sequence $\{\xi_k\}$ is monotone decreasing and converges to $x_c \in [\alpha, \beta]$, which is the nearest zero of $f(t) = c$ to $\beta$. It is an essential property that

$$\operatorname{sign} (f(t) - c) = \operatorname{sign} (f(\beta) - c) \quad (t \in (x_c, \beta)). \quad (14)$$

The new upper estimate of the global minimum on $[\alpha, \beta]$ is $a' := c$, $x_{d'} := x_c$ ($b$ unchanged). If $f(\beta) > c$, the inclusion interval $[\alpha, \beta]$ of the global minimum can be restricted to the interval $[\alpha, x_c]$, because $f(t) > c$ ($x_c < t \leq \beta$). If $f(\beta) < c$, the
inclusion interval remains $[\alpha, \beta]$ but the new upper bound $a' = f(\beta)$, $x_{a'} = \beta$, ($b$ unchanged) is better than $c$. In such a case we do not solve the equation (and save computational time).

In case (ii) we have the iterations $\xi_k < \xi_{k-1} < \cdots < \xi_1 < \xi_0$ such that either $\xi_k = \alpha$ or $\xi_k < \alpha < \xi_{k-1}$ holds. If $\xi_k < \alpha$, or $\xi_k = \alpha$ and $f(\xi_k) \neq c$, we have no solution and $\text{sign}(f(t) - c) = \text{sign}(f(\beta) - c)$ $(t \in [\alpha, \beta])$. If $f(\beta) > c$, the new upper estimate of the global minimum is $a' := a_{\text{est}} = \min \{f(\alpha), \min_{\xi_i > \alpha} f(\xi_i)\}$, $x_{a_{\text{est}}} = f(x_{a_{\text{est}}}) = a_{\text{est}}$. In case $f(\beta) < c$ the best new upper bound is

$$a := \min \left\{ f(\alpha), \min_{\xi_i > \alpha} f(\xi_i) \right\}, \quad x_a = \arg\min \left\{ f(\alpha), \min_{\xi_i > \alpha} f(\xi_i) \right\},$$

if the iterations are computed. If $f(\beta) < c$, we set the new upper bound as $a' = f(\beta)$, $x_{a'} = \beta$ and do not solve the equation.

A few of the possible situations are shown on the next figure.

Assume that alg1$_d$ is an implementation of algorithm (3) such that

$$[\alpha', \beta', a', x_{a'}, b', \text{iflag}] = \text{alg1}_d(\alpha, \beta, a, x_a, b, c)$$

denotes its application to equation $f(t) = c$ with the initial value $x_0 = \beta$. If $f(\beta) = c$, then it returns the solution $x_c = \beta$, immediately. If $f(\beta) > c$ it computes iteration (13) and sets the output values according to cases (i) or (ii). If $f(\beta) < c$, then it returns $a' = f(\beta)$ and $x_{a'} = \beta$. We may also require that

$$a \geq a' = f(x_{a'}) \geq \min_{x \in [\alpha, \beta]} f(x) > b' \geq b \quad \land \ x_{a'} \in [\alpha, \beta].$$

The iflag variable be defined by

$${\text{iflag}} = \begin{cases} 1, & \text{if } f(\beta) \geq c \land \exists x_c \in [\alpha, \beta] : f(x_c) = c \\ 0, & \text{if } f(\beta) > c \land \forall x_c \in [\alpha, \beta] : f(x_c) = c \\ -1, & \text{if } f(\beta) < c \end{cases}$$
Hence the output parameters are the following:

\[
(\alpha', \beta', a', x_{a'}, b') = \begin{cases} 
(\alpha, x_c, c, x_c, b) & \text{iflag} = 1 \\
(\alpha, \beta, a_{est}, x_{a_{est}}, c) & \text{iflag} = 0 \\
(\alpha, \beta, f(\beta), \beta, b) & \text{iflag} = -1 
\end{cases}
\]

Instead of \( a_{est} = \min \{ f(\alpha), \min_{\xi > \alpha} f(\xi) \} \) we can take \( a_{est} = f(\beta), f(\alpha) \) or any function value at a randomly taken point of \([\alpha, \beta]\). Note that \( \alpha \) never changes, \( a \) and \( x_a \) have no roles in the computations (except for the selection of \( c \)), the output \( a' \) and \( x_{a'} \) are extracted from the computed function values \( f(x) \).

Next we investigate the case, when we halve the interval \([\alpha, \beta]\) and apply alg1d to both subintervals \([\alpha, \gamma]\) and \([\gamma, \beta]\) (we assume that \( \gamma = (\alpha + \beta)/2 \)). Consider the possible situations (for simplicity, we assume that \( x_a \in [\gamma, \beta] \)):

<table>
<thead>
<tr>
<th>( x \in [\alpha, \gamma] )</th>
<th>( x \in [\gamma, \beta] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \min_{x \in [\alpha, \gamma]} f(x) &gt; a )</td>
<td>( \min_{x \in [\gamma, \beta]} f(x) \geq a )</td>
</tr>
<tr>
<td>( c &lt; \min_{x \in [\alpha, \gamma]} f(x) \leq a )</td>
<td>( c &lt; \min_{x \in [\gamma, \beta]} f(x) \leq a )</td>
</tr>
<tr>
<td>( \min_{x \in [\alpha, \gamma]} f(x) = c )</td>
<td>( \min_{x \in [\gamma, \beta]} f(x) = c )</td>
</tr>
<tr>
<td>( \min_{x \in [\alpha, \gamma]} f(x) &lt; c )</td>
<td>( \min_{x \in [\gamma, \beta]} f(x) &lt; c )</td>
</tr>
</tbody>
</table>

There are altogether 16 possible cases. Some possible situations are shown in the next figure for \( c = (a + b)/2 \).

Assume now that \((\alpha, \beta, a, x_a, b)\) is given (or popped from a stack) and we have an upper estimate \( a_{est} \) (and \( x_{a_{est}} \)) of \( \min_{x \in [l, u]} f(x) \). Estimate \( a_{est} \) is assumed to be the smallest among the upper estimates contained in the stack.

If \( a_{est} \leq b \), then we can delete \((\alpha, \beta, a, x_a, b)\) from the stack. Otherwise \( b < a_{est} \leq a \)
holds. Then we halve the interval \([\alpha, \beta]\) and apply alg1\(_d\) to both subintervals as follows.

**Algorithm 2**

1. Set the estimates \(a_{est} = f(u) (x_{a_{est}} = u), b, \) and push \((l, u, f(u), u, b)\) onto the (empty) stack.
2. **While** stack is nonempty

   - pop \((a, \beta, a, x_a, b)\) from the stack
     - if \(a_{est} \leq b\) delete \((a, \beta, a, x_a, b)\) from the stack
     - if \(a' < a_{est}\) then \(a_{est} = a', x_{a_{est}} = x_{a'}\)
     - push \((a, \gamma', a', x_{a'}, b')\) onto the stack.

   - pop \((\alpha + \beta/2, \beta', a', x_{a'}, b', iflag)\) from the stack
     - if \(a' < a_{est}\) then \(a_{est} = a', x_{a_{est}} = x_{a'}\)
     - push \((\alpha + \beta/2, \beta', a', x_{a'}, b')\) onto the stack.

**endwhile**

In the practical implementation of Algorithm 2 we used an additional condition \((\beta - \alpha < tol\) and \(a - b < tol\)) for dropping a stack element. There are many possibilities for choosing \(c_l\) and \(c_r\). For simplicity, we selected \(c_l = f((\alpha + \beta)/2 + b)/2\) and \(c_r = (f(\beta) + b)/2\) in the numerical testing.

Molinaro, Sergeyev [30], Sergeyev [33] and Kvasov, Sergeyev [24] investigated the following problem. One must check if a point \(x^*\) exists such that

\[
g(x^*) = 0, \quad g(x) > 0, \quad x \in [a, x^*] \cup (x^*, b].
\]  

These authors suggested the use of Piyavskii type global minimization algorithms to solve the problem in case of Lipschitz functions. However a direct application of algorithms (3)-(4) may also give a satisfactory answer to the problem.

1. Apply algorithm (3) with \(x_0 = b\).
2. If a zero \(\xi\) of \(g\) is found in \((a, b)\), then apply algorithm (4) with \(y_0 = a\).
3. If the first zero \(\zeta\) found by (4) is equal to \(\xi\) then the problem is solved. If \(\zeta < \xi\), the answer is negative.
4 Numerical experiments

The performance of global Lipschitz optimization clearly depends on the estimation of the unknown Lipschitz constant. Estimates of the Lipschitz constant were suggested and/or analyzed by Strongin [39], [40] Hansen, Jaumard, Lu [16], Wood, Zhang [51] and many others (see, e.g. [29], [24]). Preliminary testing indicated that none of the suggested algorithms performed well, probably due to the local character of the applied equation solver. Instead we used the following although more expensive estimate

\[ L \approx L_{est} = k \max_{i \leq n} \left\{ \frac{|f(x_i + h) - f(x_i - h)|}{2h} \right\} + d \quad (h \approx \sqrt{\epsilon_{machine}}) \]

with the values \( K = 8 \) and \( d = 1 \). Here \( \frac{|f(x_i + h) - f(x_i - h)|}{2h} \) is a second order estimate of the first derivative at the point \( x_i \), if \( f \) is differentiable three times and it is optimal in the presence of round-off error.

We used the test problem set of Hansen, Jaumard, Lu [18] numbered as 1–20, four additional functions numbered as 21–24, namely,

\[ f(x) = e^{-x} \sin\left(\frac{1}{x}\right) \quad (x \in [10^{-5}, 1]), \]

\[ f(x) = \sin x \quad (x \in [0, 1000]), \]

the Shekel function ([53])

\[ f(x) = -\sum_{i=1}^{10} \frac{1}{(k_i(x-a_i))^2 + c_i} \quad (x \in [0, 10]) \]

with parameters

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>1.5</td>
<td>2</td>
<td>3.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( c_i )</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
<td>0.3</td>
<td>0.7</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

and the Griewank function

\[ f(x) = 1 + \frac{1}{4000} x^\top x - \cos x \quad (x \in [-600, 600]). \]

In addition, we took 22 test problems of Famularo, Sergeyev, Pugliese [10] without the constraints. This test problems were numbered as 25–46.

All programs were written and tested in Matlab version R2010a (64 bit) on an Intel Core I5 PC with 64 bit Windows. We measured the achieved precision and the computational time for three different exit tolerances (10^{-3}, 10^{-5}, 10^{-7}). Algorithm 2 was compared with a Matlab implementation of the GLOBAL method of Csendes [6], Csendes, Pál, Sendín, Banga [7]. The GLOBAL method is a well-established
and maintained stochastic algorithm for multivariable functions that is based on the ideas of Boender et al [5]. The GLOBAL program can be downloaded from the web site

http://www.inf.u-szeged.hu/~csendes/index_en.html

The following table contains the averages of output errors for different exit or input tolerances.

<table>
<thead>
<tr>
<th>Tolerance</th>
<th>Algorithm 2</th>
<th>GLOBAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \times 10^{-3}$</td>
<td>$8.2343e-007$</td>
<td>$0.0088247$</td>
</tr>
<tr>
<td>$1 \times 10^{-5}$</td>
<td>$3.2244e-008$</td>
<td>$0.0039257$</td>
</tr>
<tr>
<td>$1 \times 10^{-7}$</td>
<td>$2.8846e-008$</td>
<td>$0.0020635$</td>
</tr>
</tbody>
</table>

The average execution times in [sec] are given in the next table:

<table>
<thead>
<tr>
<th>Tolerance</th>
<th>Algorithm 2</th>
<th>GLOBAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \times 10^{-3}$</td>
<td>$0.42863$</td>
<td>$0.0093795$</td>
</tr>
<tr>
<td>$1 \times 10^{-5}$</td>
<td>$2.027$</td>
<td>$0.010489$</td>
</tr>
<tr>
<td>$1 \times 10^{-7}$</td>
<td>$16.6617$</td>
<td>$0.020512$</td>
</tr>
</tbody>
</table>

It is clear that Algorithm 2 has better precision, while GLOBAL is definitely faster. The exit tolerance $1 \times 10^{-7}$ does not give essentially better precision, while the computational time significantly increased in the case of both algorithms.

The following two figures show particular details of the achieved precision and computational time.
Absolute errors

CPU time

The plots are semi-logarithmic. Hence the missing values of the first figure indicate zero output errors for both algorithms. Considering the obtained precision per CPU time we obtain the following plot.
The latter plot indicates that Algorithm 2 has a better precision rate per time unit in spite of the fact, that GLOBAL is definitely faster. Upon the basis of the presented numerical testing we conclude that Algorithm 2 might be competitive in univariate global optimization.

References


