A new method for minimization of real Lipschitz functions

J. Abaffy
John von Neumann Faculty of Informatics
Öbuda University
1034 Budapest, Bécsi u. 96/b, Hungary
Email: abaffy.jozsef@nik.uni-obuda.hu

A. Galántai
John von Neumann Faculty of Informatics
Öbuda University
1034 Budapest, Bécsi u. 96/b, Hungary
Email: galantai.aurel@nik.uni-obuda.hu

Abstract—We develop and analyze a new algorithm for global minimization of real Lipschitz functions. The proposed method uses a double bisection strategy and an always convergent solver of nonlinear equations. The computer implementation and performance are investigated in detail.

I. INTRODUCTION

In paper [2] we defined a branch and bound method to find the global minimum of the problem $f(z) \to \min (l \leq z \leq u)$, where $f : \mathbb{R}^n \to \mathbb{R}$ is sufficiently smooth and $l, u \in \mathbb{R}^n$. The suggested algorithm uses a local minimization method $z_{output} = \text{alg min} (f,z_{input})$ and a nonlinear solver $[z_{sol}, iflag] = \text{equation solve} (f, c)$ for solving the equation $f(z) = c \in \mathbb{R}$. We assume that the minimization algorithm satisfies $f(z_{output}) \leq f(z_{input})$, for any $z_{input}$. We also assume that $iflag = 1$, if a true solution $z_{sol}$ exists (that is $f(z_{sol}) = c$), and $iflag = -1$, otherwise. Denote by $f_{min}$ the global minimum of $f$, and by $B_{lower} \in \mathbb{R}$ any lower bound of $f$ such that $f_{min} \geq B_{lower}$. Let $z_0 \in D_f$ be any initial approximation to the global minimum point $(f(z_0) \geq B_{lower})$. The suggested algorithm of [2] then takes the form:

Algorithm 1

1. $z_1 = \text{alg min} (f, z_0)$
2. $a_1 = f(z_1), b_1 = B_{lower}, i = 1$

while $a_i - b_i > \text{tol}$

1. $c_i = (a_i + b_i)/2$
2. $[f, iflag] = \text{equation solve} (f, c_i)$

if $iflag = 1$

1. $z_{i+1} = \text{alg min} (f, \xi), a_{i+1} = f(z_{i+1}), b_{i+1} = b_i$

else

1. $z_{i+1} = z_i, a_{i+1} = a_i, b_{i+1} = c_i$

end

$i = i + 1$
endwhile

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]). One can prove the following [2].

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

Here we investigate an accelerated implementation of Algorithm 1 for univariate real Lipschitz functions. The global minimization of such functions has a rich literature (see e.g. Hansen, Jaumard, Lu [7], [9], [10], Pintéř [14]).

The outline of paper is the following. We develop and analyze the equation solver in Section 2. In Section 3 we develop an 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.

II. AN ALWAYS CONVERGENT SOLVER FOR REAL EQUATIONS

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

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 [17], [18] and Várterész [19] 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 [13] 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 [17], [18]) 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: 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

$$x_{n+1} = x_n - \frac{g(x_n)}{M} \ (n = 0, 1, \ldots) \quad (1)$$
either converges to the zero of \( g \) that is nearest left to \( x_0 \) or the sequence \( \{x_n\} \) exits the interval \([\alpha, \beta]\).

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

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

where \( (\xi_n) \in [0, M] \). 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 \( |g|_{\min} = \min_{x_0 \leq x \leq \alpha} |g(x)| \). 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 (1) leaves the interval in at most \( \frac{M(x_0 - \alpha)}{|g|_{\min}} \) steps. If \( g(t) \) has a zero \( \gamma \in [\alpha, x_0] \), the monotone convergence of \( \{x_n\} \) implies the relation \( |x_{n+1} - x_n| \leq |x_n - \gamma| \). Note that \( |x_{n+1} - x_n| \) is a lower estimate of the approximation error.

### III. The One-Dimensional Optimization Algorithm

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

\[
f(t) \to \min \quad \{ l \leq t \leq u, \quad f : \mathbb{R} \to \mathbb{R}, \quad l, u \in \mathbb{R}\}
\]

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 \([l, u]\).

The basic element of the modified algorithm is the solution of equation \( g(x) = f(x) - c = 0 \) on any subinterval \([\alpha, \beta] \subseteq [l, u]\). Assume that the upper and lower bounds

\[
a = f(x_a) \geq \min_{x \in [\alpha, \beta]} f(x) > b \quad (x_a \in [\alpha, \beta])
\]

given and \( c \in (a, b) \). If equation \( f(x) = c \) has a solution in \([\alpha, \beta] \), then \( \min_{x \in [\alpha, \beta]} f(x) \leq c \) and \( a \geq c \). Otherwise, if \( f(\beta) \neq c \), then we compute iterations \( \xi_0 = \beta \) and

\[
\xi_{i+1} = \xi_i - \frac{|g(\xi_i) - c|}{M} \quad (i \geq 0). \tag{2}
\]

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

\[
\text{sign} (f(t) - c) = \text{sign} (f(\beta) - c) \quad (t \in (x_c, \beta)).
\]

The new upper estimate of the global minimum on \([\alpha, \beta]\) is \( a' := c, x_{a'} := x_c \) (b unchanged). If \( f(\beta) > c \), the inclusion interval \([\alpha, x_c]\) 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 \( a' = f(\beta), x_{a'} = \beta \) \((b \text{ unchanged})\) 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 sign \( f(t) - c) = \text{sign} (f(\beta) - c) \quad (t \in [\alpha, \beta]) \). If \( f(\beta) > c \), the new upper estimate of the global minimum is \( a' := a_{\text{est}} = \min \{ f(\alpha), \min_{\xi_i > \alpha} f(\xi_i) \} \). In case \( f(\beta) < c \) the best new upper bound is

\[
a := \min \left\{ f(\alpha), \min_{\xi_i > \alpha} f(\xi_i) \right\},
\]

\[
x_a = \arg \min \left\{ f(\alpha), \min_{\xi_i > \alpha} f(\xi_i) \right\},
\]

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.

Assume that \( \text{alg}_d \) is an implementation of algorithm (1) such that

\[
[a', b', x_{a'}, b, i\text{flag}] = \text{alg}_d (\alpha, \beta, a, x_a, b; c)
\]

denotes its application to equation \( f(t) = c \) with the initial value \( x_0 = 0 \). If \( f(\beta) = c \), then it returns the solution \( x_c = \beta \), immediately. If \( f(\beta) > c \) it computes iteration (2) and sets the output values according to cases (i) or (ii). If \( f(\beta) < c \), then it returns \( a' = f(\beta) \) and \( x_{a'} = \beta \). The \( i\text{flag} \) variable is defined by

\[
i\text{flag} = \begin{cases} 1, & \text{if } f(\beta) \geq c \land \exists x_c \in [\alpha, \beta] : f(x_c) = c \land \text{alg}_d (\alpha, \beta, a, x_a, b; c) \\ 0, & \text{if } f(\beta) > c \land \exists x_c \in [\alpha, \beta] : f(x_c) = c \land \text{alg}_d (\alpha, \beta, a, x_a, b; c) \\ -1, & \text{if } f(\beta) < c \land \exists x_c \in [\alpha, \beta] : f(x_c) = c \land \text{alg}_d (\alpha, \beta, a, x_a, b; c) \end{cases}
\]

Hence the output parameters are the following:

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

Instead of \( a_{\text{est}} = \min \{ f(\alpha), \min_{\xi_i > \alpha} f(\xi_i) \} \) we can take \( a_{\text{est}} = f(\beta), f(\alpha) \) 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 \( f(\xi_i) \).

In order to accelerate Algorithm 1 we halve the \( t \)-interval and use a stack to include or drop subintervals. Assume that \( (\alpha, \beta, a, x_a, b) \) is given (or popped from a stack) and we have an upper estimate \( a_{\text{est}} \) (and \( x_{a_{\text{est}}} \)) of \( \min_{x \in [l, u]} f(x) \). Estimate \( a_{\text{est}} \) is assumed to be the smallest among the upper estimates contained in the stack.

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

**Algorithm 2**
Set the estimates $a_{ext} = f(u)$ ($x_{a_{ext}} = u$), $b$, and push $(l, u, f(u), u, b)$ onto the (empty) stack.

while stack is nonempty

pop $(a, \beta, a, x_a, b)$ from the stack

if $a_{ext} \leq b$ delete $(a, \beta, a, x_a, b)$ from the stack

$[\alpha, \gamma', a', x_{a'}, b', i, flag] = \text{alg1d} (\alpha, x_{a}, b; c_i)$

if $a' < a_{ext}$ then $a_{ext} = a'_i$, $x_{a_{ext}} = x_{a'_i}$
  push $(\alpha, \gamma', a', x_{a'}, b')$ onto the stack.

$[\alpha+\beta, \beta', a', x_{a'}, b', i, flag] = \text{alg1d} (\alpha, x_{a}, b; c_r)$

if $a' < a_{ext}$ then $a_{ext} = a'_r$, $x_{a_{ext}} = x_{a'_r}$
  push $(\alpha+\beta, \beta', a', x_{a'}, b')$ 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 $c_l = \left( f\left(\frac{\alpha+\beta}{2}\right) + b\right) / 2$ and $c_r = (f(\beta) + b)/2$ in the numerical testing.

IV. 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 [15], [16] Hansen, Jaumard, Lu [8], Wood, Zhang [20] and many others (see, e.g. [12], [11]). 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

$$L \approx L_{\max}^{ext} = k \max_{i < n} \left\{ \frac{|f(x_i + h) - f(x_i - h)|}{2h} \right\} + d$$

with the values $k = 8$, $d = 1$ and $h \approx \sqrt{\epsilon_{\text{machine}}}$. 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 [10] numbered as 1–20, four additional functions numbered as 21–24, namely,

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

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

the Shekel function

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

with parameters of [21], and the Griewank function

$$f(x) = 1 + \frac{1}{4000} x^2 - \cos x \quad (x \in [-600, 600])$$

In addition, we took 22 test problems of Famularo, Sergeyev, Pugliese [6] 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 [4], Csendes, Pál, Sendín, Banga [5]. The GLOBAL method is a well-established and maintained stochastic algorithm for multivariable functions that is based on the ideas of Boender et al [3]. 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>Index</th>
<th>Algorithm 2</th>
<th>GLOBAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1e – 3</td>
<td>8.2343e – 007</td>
<td>0.0088247</td>
</tr>
<tr>
<td>1e – 5</td>
<td>3.2244e – 008</td>
<td>0.0039257</td>
</tr>
<tr>
<td>1e – 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>Index</th>
<th>Algorithm 2</th>
<th>GLOBAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1e – 3</td>
<td>0.42863</td>
<td>0.0093795</td>
</tr>
<tr>
<td>1e – 5</td>
<td>2.027</td>
<td>0.010489</td>
</tr>
<tr>
<td>1e – 7</td>
<td>16.601</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 $1e – 7$ does not give essentially better precision, while the computational time significantly increased in the case of both algorithms.

Figures 1, and 2 show particular details of the achieved precision and computational 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 plot in Fig. 3.
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