Electrical, Control and Communication Engineering
ISSN 2255-9159 (online)
ISSN 2255-9140 (print)
2022, vol. 18, no. 1, pp. 27–36
https://doi.org/10.2478/ecce-2022-0004
https://content.sciendo.com

Three-Point Iterated Interval Half-Cutting for Finding All Local Minima of Unknown Single-Variable Function

Vadim Romanuke* (*Vinnytsia Institute of Trade and Economics of State University of Trade and Economics, Vinnytsia, Ukraine*)

Abstract **– A numerical method is suggested to find all local minima and the global minimum of an unknown single-variable function bounded on a given interval regardless of the interval length. The method has six inputs: three inputs defined straightforwardly and three inputs, which are adjustable. The endpoints of the initial interval and a formula for evaluating the single-variable function at any point of this interval are the straightforward inputs. The three adjustable inputs are a tolerance with the minimal and maximal numbers of subintervals. The tolerance is the secondary adjustable input. Having broken the initial interval into a set of subintervals, the three-point iterated half-cutting "gropes" around every local minimum by successively cutting off a half of the subinterval or dividing the subinterval in two. A range of subinterval sets defined by the minimal and maximal numbers of subintervals is covered by running the three-point half-cutting on every set of subintervals. As a set of values of currently found local minima points changes less than by the tolerance, the set of local minimum points and the respective set of function values at these points are returned. The presented approach is applicable to whichever task of finding local extrema is. If primarily the purpose is to find all local maxima or the global maximum of the function, the presented approach is applied to the function taken with the negative sign. The presented approach is a significant and important contribution to the field of numerical estimation and approximate analysis. Although the method does not assure obtaining all local minima (or maxima) for any function, setting appropriate minimal and maximal numbers of subintervals makes missing some minima (or maxima) very unlikely.**

Keywords **– Finding extrema, interval half-cutting, local minima, subintervals, unknown single-variable function.**

I. Practical Issues of Finding a Minimum

Finding minima of a function whose equation is known is an easy academic task that can be fulfilled either by algebraic (symbolic) or numerical methods [1], [2]. In most practical problems, the equation of a function is unknown, so its derivatives are not available and neither a symbolic approach nor a numerical method is applicable [3], [4]. Another practical issue arises when a symbolic (exact) approach cannot be applied to a known function equation containing, e. g., analytically non-differentiable parts [5], [6], and numerical methods are the way to minimize approximately, but computing function values either takes unreasonably long time or is too expensive [7], [8]. Thus, no numerical method is applicable as there is no object (i. e., tabulated function) to which it might be applied.

The most prominent examples of the inapplicability of the exact and numerical methods are the problem of fine-tuning hyperparameters and training parameters of neural networks [9], [10], continuous and discrete parameter adjustment for various algorithms [8], [11], [12], the problem of optimizing the phased array size and parameters of beamforming [13], [14], etc. In such examples, an objective function is unknown and its evaluation is either time-consuming or resource-consuming, or both (just like the cases of neural networks and phased arrays). The fact of the objective function is defined on a (known) discrete set (e. g., searching for an optimal size) does not matter. Using a numerical method always implies that the function is considered a finite set of values corresponding to a finite set of points to which those values are assigned.

The existing methods of finding minimum without involving the single-variable function equation and function tiny-step evaluation rely on successively narrowing the range of values on the specified interval, which makes it relatively slow, but very robust [15], 16]. The method of golden-section search maintains the function values for four points whose three interval widths are in the golden ratio [17], [18]. These ratios are maintained for each iteration and are maximally efficient. For a strictly unimodal function with an extremum inside the interval, the golden-section search will find that extremum, while for an interval containing multiple extrema (possibly including the interval boundaries), it will converge to one of them. This is an obvious demerit of the method because one cannot be sure that an interval contains a single minimum and so other minima are just omitted (Fig. 1).

The Fibonacci search technique is a similar algorithm to find the extremum (minimum or maximum) of a finite sequence of values that has a single local minimum or local maximum [19], [20]. The algorithm maintains a bracketing of the solution in which the length of the bracketed interval is a Fibonacci number. As the Fibonacci search technique is derived from the golden-section search, it also may fail in determining the global minimum (similarly to that shown in Fig. 1).

* Corresponding author.
E-mail: romanukevadimv@gmail.com

Another technique for finding the minimum of a unimodal function is the ternary search algorithm [21], [22]. A ternary search determines either that the minimum cannot be in the first third of the domain or that it cannot be in the last third of the domain, and then repeats on the remaining two thirds. The ternary search is slightly faster that the golden-section search, but it also may omit the global minimum (in the example in Fig. 1, the ternary search finds the local minimum and omits the global minimum).

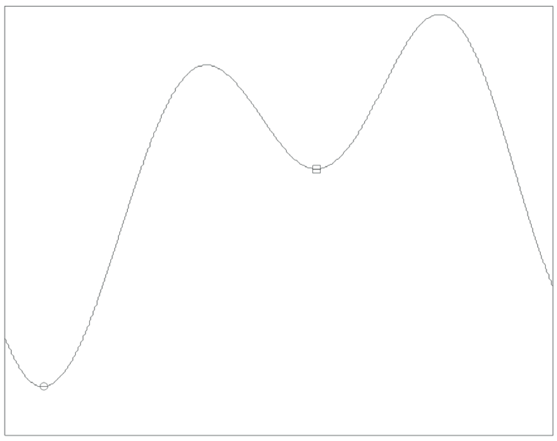

Fig. 1. An example of when the golden-section search finds a local minimum (the squared point to the right), whereas the global minimum (the circled point to the left) is omitted.

The global minimum can be found by the genetic algorithm [23], [24]. The genetic algorithm based on imitating a biological evolution with selection and crossover of solution candidates is far more reliable in determining the minimum. For instance, the genetic algorithm finds the global minimum in Fig. 1, and, in a relatively more complex example, finds the global minimum in Fig. 2. However, often even the genetic algorithm fails to find the global minimum (Fig. 3). Besides, the genetic algorithm is far slower than both the golden-section search and ternary search algorithms as it must operate on rather a great deal of solution candidates. Therefore, if the function evaluation is expensive (in the sense of either computational time or a factual cost to evaluate a value of the function), the genetic algorithm is practically inappropriate (unacceptable).

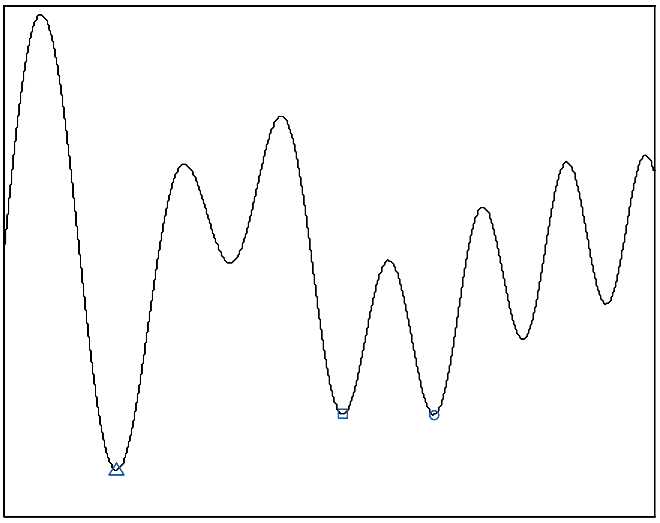

Fig. 2. An example of when both the golden-section search and ternary search find different local minima (the squared point minimum is found by the golden-section search and the circled point minimum is found by the ternary search), whereas the global minimum (the triangled point to the left) is found only by the genetic algorithm.

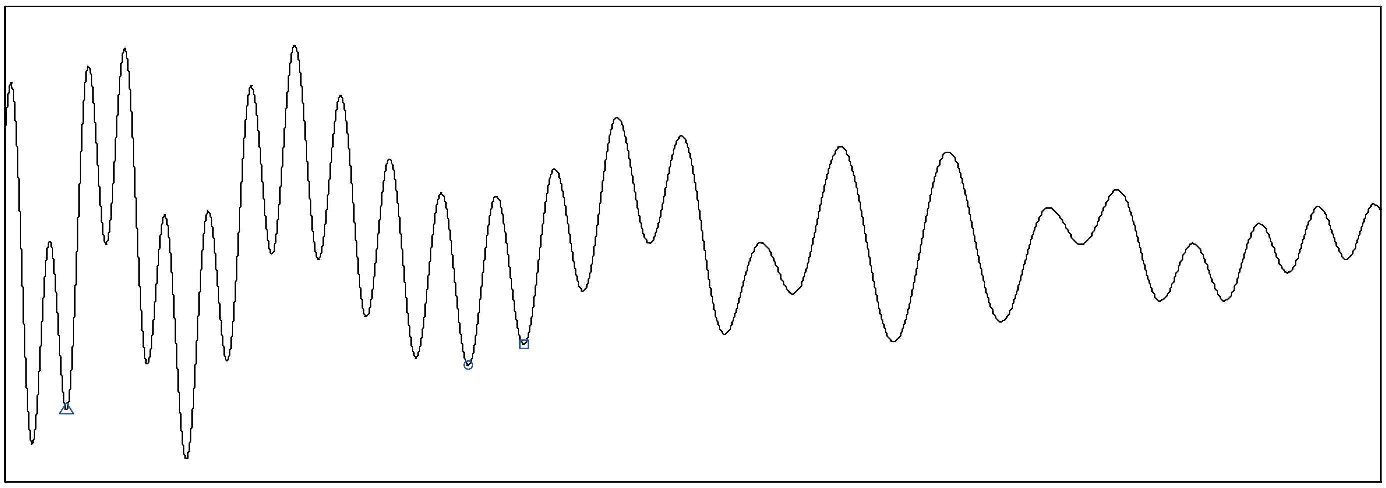

Fig. 3. An example of when both the ternary search and golden-section search find different local minima (the circled and squared points, respectively), and the genetic algorithm fails to find the global minimum (considering the function values, a local minimum found by the genetic algorithm and marked as the triangled point is pretty close to the global minimum but still it is not the global minimum; moreover, there is another local minimum which is even closer to the global minimum).

Apart from the need to determine the global minimum, often all local minima of an unknown function are required to be found. However, the mentioned methods allow determining (or locating) only one minimum on an interval. Meanwhile, the interval may contain other local minima among which the global minimum may be. To find all local minima on an interval, this interval should be broken into a subset of subintervals, on each of which no more than a single local minimum is supposed to be. Obviously, this supposition is not always true, so some local minima including the global one may be lost.

II. Goal and Objectives to Be Accomplished

As the existing approaches are incapable of determining all local minima of an unknown single-variable function within any bounded interval, the goal is to develop a method by which they could be efficiently found regardless of the interval length. For this reason, the interval should be broken into narrower subintervals. To achieve the goal, the following objectives are to be accomplished:

1. To impose specific conditions on the function and its extrema on the bounded interval.
2. To suggest a method (algorithm #1) of determining a local minimum on the interval or returning a specific answer implying that the interval, apart from the local minimum, contains other extrema.
3. To suggest a method (algorithm #2) of breaking the initial interval into a set of subintervals, for each of which a local minimum is found or the specific answer is returned by algorithm #1. Thus, algorithm #1 is to be incorporated into algorithm #2.
4. To suggest a method (algorithm #3) of adjusting algorithm #2 so that all the local minima would be determined with an acceptable accuracy (tolerance) and no specific answer would be returned by algorithm #1. Algorithm #3 should incorporate algorithm #2 to be a novel approach of finding local minima of a single-variable function.
5. To exemplify the suggested approach.
6. To discuss applicability and significance of the approach.
7. To make an appropriate conclusion on it.

III. Function and Its Extrema on a Bounded Interval

It is assumed that a function $f(x)$ defined on interval $[a; b]$ by $b > a$ is bounded and not a piecewise constant function. Whether function $f(x)$ is continuous on $[a; b]$ or not, this function is supposed to have no points of jump discontinuity. In addition, it is supposed that the function has an extremum on open interval $(a; b)$. However, this supposition may be false and so the function is either increasing or decreasing on interval $[a; b]$. The supposition about that the function is strictly unimodal with an extremum inside the interval is not made.

In further consideration, referring to function $f(x)$ implies calling to an external routine to calculate (or evaluate) its value at point x rather than to the equation of the function. This is so because function $f(x)$ is assumed to be unknown as it happens in overwhelming majority of practical tasks.

IV. Three-Point Interval Half-Cutting

Why two points are insufficient to be probed in searching for a minimum is illustrated in Fig. 4. Therefore, the case with three points must be studied. Let these points x_1, x_2, x_3 be selected uniformly within interval $[a; b]$:

$$x_1 = a + \frac{b-a}{4} = \frac{3a+b}{4},$$
$$x_2 = x_1 + \frac{b-a}{4} = a + \frac{b-a}{2} = \frac{a+b}{2},$$
$$x_3 = x_2 + \frac{b-a}{4} = a + \frac{3\cdot(b-a)}{4} = \frac{a+3b}{4}. \tag{1}$$

Alternatively, points (1) are calculated starting from the middle point:

$$x_2 = \frac{a+b}{2}, \quad x_1 = \frac{a+x_2}{2}, \quad x_3 = \frac{x_2+b}{2}. \tag{2}$$

There are four inputs to algorithm #1: the interval endpoints a, b, a formula for evaluating function $f(x)$ at any point x, and tolerance ε (a sufficiently small positive number).

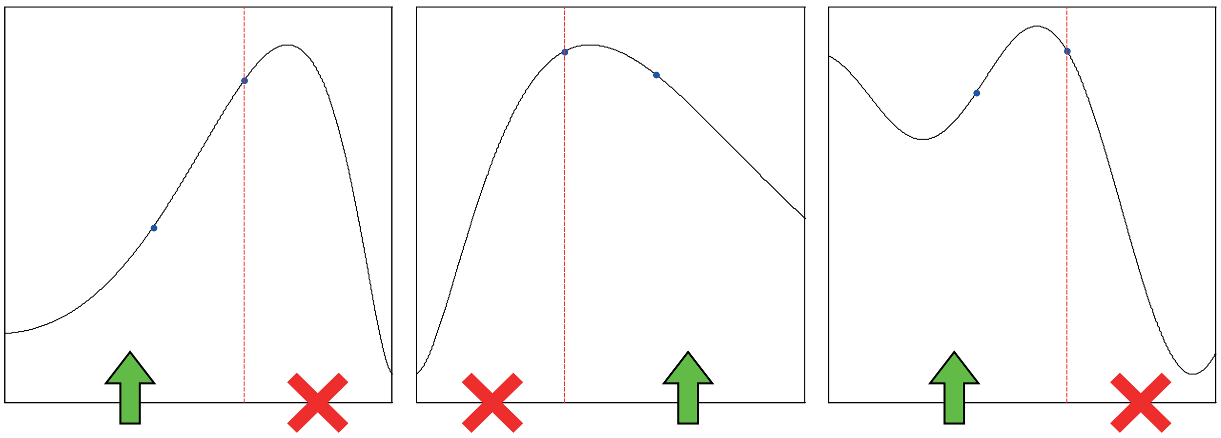

Fig. 4. Examples of when probing only two internal points (using the golden-section search in this case) leads to cutting the subinterval containing the least value of a function on a given interval.

At the first (initial) step of algorithm #1, the function is evaluated at points (2) and the minimum of values

$$f(x_1), \quad f(x_2), \quad f(x_3) \tag{3}$$

is found. Denote this minimum by y_*. Point $x_* \in \{x_1, x_2, x_3\}$ at which this minimum is found is stored. Minimum value $y_* = f(x_*)$ is stored as well. Here, $\Delta x = x_2 - x_1$.

While $\Delta x > \varepsilon$, the following routine is executed. If

$$f(x_1) \geqslant f(x_2) \geqslant f(x_3), \quad (4)$$

then

$$a = x_2 \quad (5)$$

and (2) are re-specified. If

$$f(x_1) \leqslant f(x_2) \leqslant f(x_3), \quad (6)$$

then

$$b = x_2 \quad (7)$$

and (2) are re-specified. If

$$\min\{f(x_1), f(x_2), f(x_3)\} = f(x_2), \quad (8)$$

then

$$a = x_1, \quad b = x_3 \quad (9)$$

and (2) are re-specified. If

$$\max\{f(x_1), f(x_2), f(x_3)\} = f(x_2), \quad (10)$$

then algorithm #1 stops returning the specific answer implying that the interval, apart from the local minimum, contains other extrema. This answer is returned in the form of the empty set of function minimum ($M_* = \varnothing$) and two halves [a; x_2] and [x_2; b]. Otherwise, if (10) is false, the minimum of values (3) is found, values $\{x_*, y_*\}$ are stored, and new $\Delta x = x_2 - x_1$ is calculated.

All the four cases of the routine conditions and their outcomes are illustrated in Fig. 5. Unlike using the golden-section search, by which the interval is narrowed by $\frac{\sqrt{5}+1}{2} \approx 1.618$ times, the interval by algorithm #1 becomes twice as narrowed.

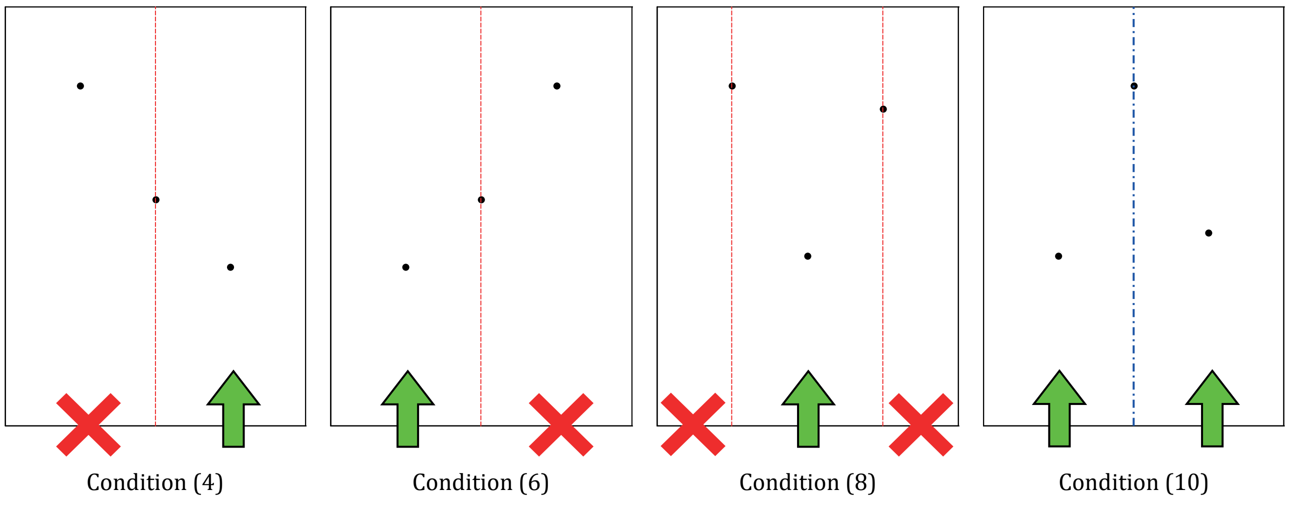

Fig. 5. The four possible cases and their outcomes issued from one of conditions (4), (6), (8), (10).

In fact, algorithm #1 is a three-point interval half-cutting running, while $\Delta x > \varepsilon$ unless condition (10) turns true. The returned output is either $M_* = \{x_*, y_*\}$ or $M_* = \varnothing$ and two halves (subintervals) [a; x_2] and [x_2; b]. The flowsheet of algorithm #1 whose computational complexity is similar to that of the golden-section search [17] is as follows:

```
1. Input $a$, $b$, $f(x)$, $\varepsilon$.
2. Specify points (2).
3. Evaluate $f(x)$ at points (2).
4. Find the minimum $y_*$ of values (3) and its minimum point $x_*$.
5. Calculate $\Delta x = x_2 - x_1$.
6. While $\Delta x > \varepsilon$ do:
    6.1. If (4) is true then assign (5) and re-specify points (2).
    6.2. If (6) is true then assign (7) and re-specify points (2).
    6.3. If (8) is true then assign (9) and re-specify points (2).
    6.4. If (10) is true then return $M_* = \varnothing$ and $[a; x_2]$, $[x_2; b]$.
    6.5. Find $\{x_*, y_*\}$ and calculate $\Delta x = x_2 - x_1$.
7. Return $M_* = \{x_*, y_*\}$.
```

Algorithm #1 itself is insufficient to find all minima of a function. Nevertheless, owing to the third point, it can sometimes outperform any two-point search (at least by not returning a "false" local minimum and simultaneously losing a "real" local minimum like in the examples in Fig. 4). For instance, algorithm #1, similarly to the golden-section search, returns the "false" local minimum and loses the "real" local minimum in the left subplot example in Fig. 4, but it does not make such a mistake for the middle and right subplot examples, for which algorithm #1 returns the two subintervals. These subintervals are to be further studied whether each contains minima.

V. Running the Three-Point Interval Half-Cutting on a Set of Subintervals

As the initial interval [*a*; *b*] may contain multiple local minima, it is reasonable to break the initial interval into a set of equal-length subintervals, for each of which a local minimum would be found or the specific answer would be returned by algorithm #1. Denote the number of subintervals by *N*. These subintervals are

$$\left\{[a_i; b_i]\right\}_{i=1}^{N} \tag{11}$$

by

$$a_1 = a\,,\ \ b_1 = a + \frac{b-a}{N}\,,\ \ b_N = b \tag{12}$$

and

$$a_i = b_{i-1} \ \text{ by } \ b_i = a_i + \frac{b-a}{N} \ \text{ for } i = \overline{2, N}\,. \tag{13}$$

There are five inputs to algorithm #2 (which will incorporate algorithm #1): the interval endpoints *a*, *b*, a formula for evaluating function *f*(*x*) at any point *x*, tolerance ε, the number of subintervals *N*. At the first step of algorithm #2, algorithm #1 is applied to subinterval $[a_i; b_i]$ for $i = \overline{1, N}$. If x_* is found for subinterval $[a_i; b_i]$, i. e. $M_* \neq \varnothing$ for this subinterval, then the condition eliminating endpoints is checked. If

$$|x_* - a_i| < \varepsilon \ \text{ or } \ |x_* - b_i| < \varepsilon \tag{14}$$

then x_* is deleted as it is too close either to the left or right endpoint of the subinterval (and, thus, the function is supposed to have no "internal" minima on this subinterval). Otherwise, x_* is stored along with $f(x_*)$ as a local minimum on subinterval $[a_i; b_i]$. If $M_* = \varnothing$ for subinterval $[a_i; b_i]$, then algorithm #1 is applied to both halves

$$\left[a_i; \frac{a_i + b_i}{2}\right] \ \text{ and } \ \left[\frac{a_i + b_i}{2}; b_i\right]. \tag{15}$$

For each of halves (15) the eliminating-endpoint condition is checked. A minimum $x_*^{(\text{left})}$, if it is found on subinterval

$$\left[a_i; \frac{a_i + b_i}{2}\right], \tag{16}$$

is deleted if

$$\left|x_*^{(\text{left})} - a_i\right| < \varepsilon \ \text{ or } \ \left|x_*^{(\text{left})} - \frac{a_i + b_i}{2}\right| < \varepsilon\,; \tag{17}$$

otherwise, $x_*^{(\text{left})}$ is stored along with $f\left(x_*^{(\text{left})}\right)$ as a local minimum on subinterval (16). A minimum $x_*^{(\text{right})}$, if it is found on subinterval

$$\left[\frac{a_i + b_i}{2}; b_i\right], \tag{18}$$

is deleted if

$$\left|x_*^{(\text{right})} - \frac{a_i + b_i}{2}\right| < \varepsilon \ \text{ or } \ \left|x_*^{(\text{right})} - b_i\right| < \varepsilon\,; \tag{19}$$

otherwise, $x_*^{(\text{right})}$ is stored along with $f\left(x_*^{(\text{right})}\right)$ as a local minimum on subinterval (18).

At the second step of algorithm #2, all the local minima found on *N* subintervals (11) are aggregated into a set X_* whose elements are sorted in ascending order. Let $Y_* = f(X_*)$ be a set of function values at the local minima in X_*. If

$$f(a) < \max f(X_*) \tag{20}$$

then point $x = a$ can be counted a local boundary minimum and

$$X_*^{(\text{obs})} = X_*\,,\ \ X_* = \{a\} \cup X_*^{(\text{obs})}\,, \tag{21}$$

$$Y_*^{(\text{obs})} = Y_*\,,\ \ Y_* = \{f(a)\} \cup Y_*^{(\text{obs})}\,. \tag{22}$$

If

$$f(b) < \max f(X_*) \tag{23}$$

then point $x = b$ can be counted a local boundary minimum and

$$X_*^{(\text{obs})} = X_*\,,\ \ X_* = X_*^{(\text{obs})} \cup \{b\}\,, \tag{24}$$

$$Y_*^{(\text{obs})} = Y_*\,,\ \ Y_* = Y_*^{(\text{obs})} \cup \{f(b)\}\,. \tag{25}$$

At the end of the routine, algorithm #2 returns sets X_* and Y_*. The flowsheet of algorithm #2 is as follows:

1. Input a, b, $f(x)$, ε, N.
2. Break interval $[a; b]$ into subintervals (11) by (12) and (13).

```
3. For $i = \overline{1, N}$ do:
    3.1. Apply algorithm #1 to subinterval $[a_i; b_i]$.
    3.2. If $M_* \neq \varnothing$ then check:
            If (14) holds then delete $x_*$;
            Else store $x_*$ and $f(x_*)$ as a local minimum on subinterval $[a_i; b_i]$.
        Else
            Apply algorithm #1 to both halves (15).
                If $M_* \neq \varnothing$ for (16) then check:
                    If (17) holds then delete $x_*^{(\text{left})}$;
                    Else store $x_*^{(\text{left})}$ and $f(x_*^{(\text{left})})$ as a local minimum on subinterval (16).
                If $M_* \neq \varnothing$ for (18) then check:
                    If (19) holds then delete $x_*^{(\text{right})}$;
                    Else store $x_*^{(\text{right})}$ and $f(x_*^{(\text{right})})$ as a local minimum on subinterval (18).
4. Aggregate all the local minima found on subintervals (11) into a set $X_*$.
5. Sort them in ascending order.
6. Create a set $Y_* = f(X_*)$.
7. If (20) holds then fulfil (21) and (22).
8. If (23) holds then fulfil (24) and (25).
9. Return sets $X_*$ and $Y_*$.
```

VI. Covering a Range of Subinterval Sets

Algorithm #2, if number *N* is sufficiently great, returns all the local minima of a function including its local boundary minima, if any, satisfying inequalities (20), (23). But how to guess the sufficiently great number of subintervals? Inputting always very great numbers of subintervals into algorithm #2 will significantly slow down finding minima. Inputting a fewer number of subintervals may lead to losing some minima. Therefore, it is reasonable to try a set of these numbers to see whether new minima appear as number *N* is increased.

There are six inputs to algorithm #3 (which will incorporate algorithm #2): the interval endpoints *a*, *b*, a formula for evaluating function *f*(*x*) at any point *x*, tolerance ε, the minimal number of subintervals $N_{\min}$, the maximal number of subintervals $N_{\max}$. At the first step of algorithm #3, algorithm #2 is applied by $N = N_{\min}$. While $M_* = \varnothing$ and $N < N_{\max}$, the number of subintervals is increased by 1 and algorithm #2 is applied again. If $M_* = \varnothing$ at $N = N_{\max}$, then

$$y_* = \min\{f(a), f(b)\} \tag{26}$$

and

$$x_* = a \text{ by } y_* = f(a) \tag{27}$$

and

$$x_* = b \text{ by } y_* = f(b), \tag{28}$$

whereupon algorithm #3 stops returning just a pair $\{x_*, y_*\}$. Otherwise, if a nonempty set X_* is found at some $N = N_1$ (i. e., $M_* \neq \varnothing$ at $N = N_1$), a counter of the number of subintervals is set at 1: *j* = 1. Besides, the found sets X_* and Y_* are stored indicating the counter:

$$X_*^{(1)} = X_*, \quad Y_*^{(1)} = Y_*. \tag{29}$$

Then, at the second step of algorithm #3, the number of subintervals is increased by 1 (while $N < N_{\max}$) and counter *j* is increased by 1, whereupon algorithm #2 is applied by this new number of subintervals N_j and

$$X_*^{(j)} = X_*, \quad Y_*^{(j)} = Y_*. \tag{30}$$

If

$$\left|X_*^{(j)}\right| = \left|X_*^{(j-1)}\right| \tag{31}$$

by

$$X_*^{(j-1)} = \left\{x_*^{(j-1,h)}\right\}_{h=1}^{H} \tag{32}$$

and

$$X_*^{(j)} = \left\{x_*^{(j,h)}\right\}_{h=1}^{H}, \tag{33}$$

then it is checked whether those *H* values in sets (32) and (33) are sufficiently close (or, being accurate to ε, are practically the same). Thus, if

$$\max_{\underline{h=1,H}} \left|x_*^{(j,h)} - x_*^{(j-1,h)}\right| < \varepsilon \tag{34}$$

then algorithm #2 is applied by the number of subintervals $2N_j$ returning a set X_* denoted by Z_*. If

$$|Z_*| = \left|X_*^{(j)}\right| \tag{35}$$

by

$$Z_* = \left\{z_*^{(h)}\right\}_{h=1}^{H}, \tag{36}$$

then it is checked whether those *H* values in sets (35) and (36) are sufficiently close. Thus, if

$$\max_{h=\overline{1,H}} \left| z_*^{(h)} - x_*^{(j,h)} \right| < \varepsilon \tag{37}$$

then algorithm #3 stops returning

$$X_* = X_*^{(j-1)} = \left\{ x_*^{(j-1,h)} \right\}_{h=1}^{H},$$
$$Y_* = Y_*^{(j-1)} = \left\{ y_*^{(j-1,h)} \right\}_{h=1}^{H} = \left\{ f\left(x_*^{(j-1,h)} \right) \right\}_{h=1}^{H}. \tag{38}$$

In fact, algorithm #3 covering a range of algorithm #2 runs is a three-point iterated interval half-cutting. It is worth noting that algorithm #3 does not return the specific answer. The specific answer is an internal object of algorithm #1 serving to divide a subinterval further. The flowsheet of algorithm #3 is as follows:

```
1. Input $a$, $b$, $f(x)$, $\varepsilon$, $N_{\min}$, $N_{\max}$.
2. Assign $N = N_{\min} - 1$.
3. While $M_* = \varnothing$ and $N < N_{\max}$ do:
   3.1. Increase $N$ by 1.
   3.2. Apply algorithm #2.
4. If $M_* = \varnothing$ then do:
   4.1. Assign (26) — (28).
   4.2. Return $M_* = \{x_*, y_*\}$.
5. Assign $j = 1$.
6. Assign (29).
7. For $N = \overline{N_1 + 1, N_{\max}}$ do:
   7.1. Increase $j$ by 1.
   7.2. Apply algorithm #2.
   7.3. Assign (30).
        If (31) holds by (32) and (33) then check:
            If (34) holds then apply algorithm #2 by
            the number of subintervals $2N_j$;
                Denote $X_*$ by $X_*$.
                If (35) holds by (36) then check:
                    If (37) holds then:
                        Decrease $N$ by 1.
                        Assign $N_* = N$.
                        Return $N$ and (38).
8. If variable $N_*$ does not exist, return null.
```

If the task is to find the global minimum, then it is fulfilled as a trivial appendix to the three-point iterated interval half-cutting. As sets (38) are obtained, the global minimum function value

$$y_{**} = \min_{h=\overline{1,H}} y_*^{(j-1,h)} \tag{39}$$

is calculated and every global minimum point

$$x_{**} \in \left\{ x_*^{(j-1,h)} \right\}_{h=1}^{H} \tag{40}$$

at which

$$y_{**} = f(x_{**}) \tag{41}$$

is extracted. If the global minimum is to be found purely on open interval (a; b), then the global minimum function value

$$y_{**} = \min\left\{ Y_*^{(j-1)} \setminus \left\{ f(a), f(b) \right\} \right\} \tag{42}$$

is calculated and global minimum point (40) at which (41) holds is extracted.

To find approximate values of all minimum points and respective minimum function values, the property of the function differentiability is unnecessary. It is sufficient that the function (regardless of whether it is unknown or known) be bounded and have a finite number of local minimum points.

Theorem 1. If a bounded function $f(x)$ has a finite number of local minimum points on interval [a; b], then $\exists\, N^*$ such that algorithm #2 returns this number of approximate local minimum points which differ from the true local minimum points at most by ε.

Proof. If the number of minima is finite, then interval [a; b] can be broken by algorithm #2 into N^* equal-length subintervals such that each of the subintervals either will contain a single minimum point without local maxima points or will not contain local minimum points at all. Then, in the first case, one of the conditions by (4), (6), (8) turns true on every step of algorithm #1 giving eventually an approximate local minimum point that differs from the true local minimum point at most by ε. In the second case, when no local minimum points are within the subinterval, one of the conditions by (4), (6), (10) turns true on every step of algorithm #1 that eventually deletes every candidate to be a local minimum point by using (17) or (19). The theorem has been proved.

VII. Examples

To compare performance of three-point iterated interval half-cutting to that of the golden-section search, ternary search, genetic algorithm, specific instances are taken that have multiple extrema [2], [3], [8]. The respective experiments confirm that the suggested method outperforms those and other approaches. An example of applying the three-point iterated interval half-cutting is shown in Fig. 6, where a toy function

$$f(x) = \sin(13.24x)\cos\left(3.36x^2\right)e^{-0.96\sin(0.32x)} \tag{43}$$

is used to model the problem of minimization on interval [–2.15; 1.1]. The 12 local minima (including the global minimum) are found by $N = 9$, i. e. by breaking the initial interval just into 9 subintervals. Another, seemingly a more difficult example, is presented in Fig. 7 for a toy function

$$f(x) = \cos\left(x\cos(2x)e^{-0.05\sin(4x)} - \frac{\pi}{7} \right) e^{0.08x\cos(x)\sin(0.5x)} \tag{44}$$

minimized on interval [−18.8; 4], where the function (surely, unbeknown to a researcher in reality) has a sort of modulation. The 48 local minima (including the global minimum) are found here by $N = 77$ (by breaking the initial interval into 77 subintervals). Fig. 8 shows the result of applying the three-point iterated interval half-cutting to find all the local minima of a toy function

$$f(x) = \cos\left(\sin(20x)\cos(8x^3)e^{-0.25\sin(x)} - \frac{\pi}{7}\right) - $$

$$- 2\sin\left(\sin(5x^2)\cos(6x)e^{-1.2\sin(2x)} + \frac{\pi}{6}\right) \tag{45}$$

on interval [3; 3.6]. Compared to the example in Fig. 7 with function (44), the example in Fig. 8 with function (45) can be thought of as a more computationally expensive: the algorithm takes here 116 subintervals (by 50.65 % greater than that in Fig. 7) to find all the 41 local minima (by 14.58 % less than that in Fig. 7).

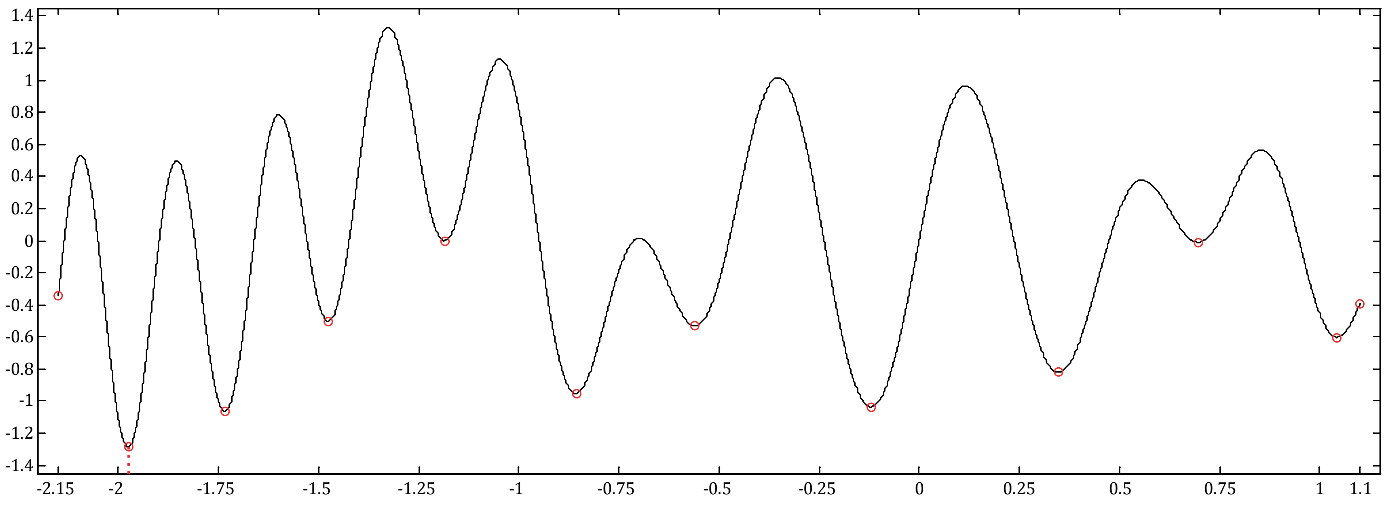

Fig. 6. The 12 local minima, including both the local boundary minima and the global minimum (marked by the dotted line downward) of function (43) on interval [−2.15; 1.1].

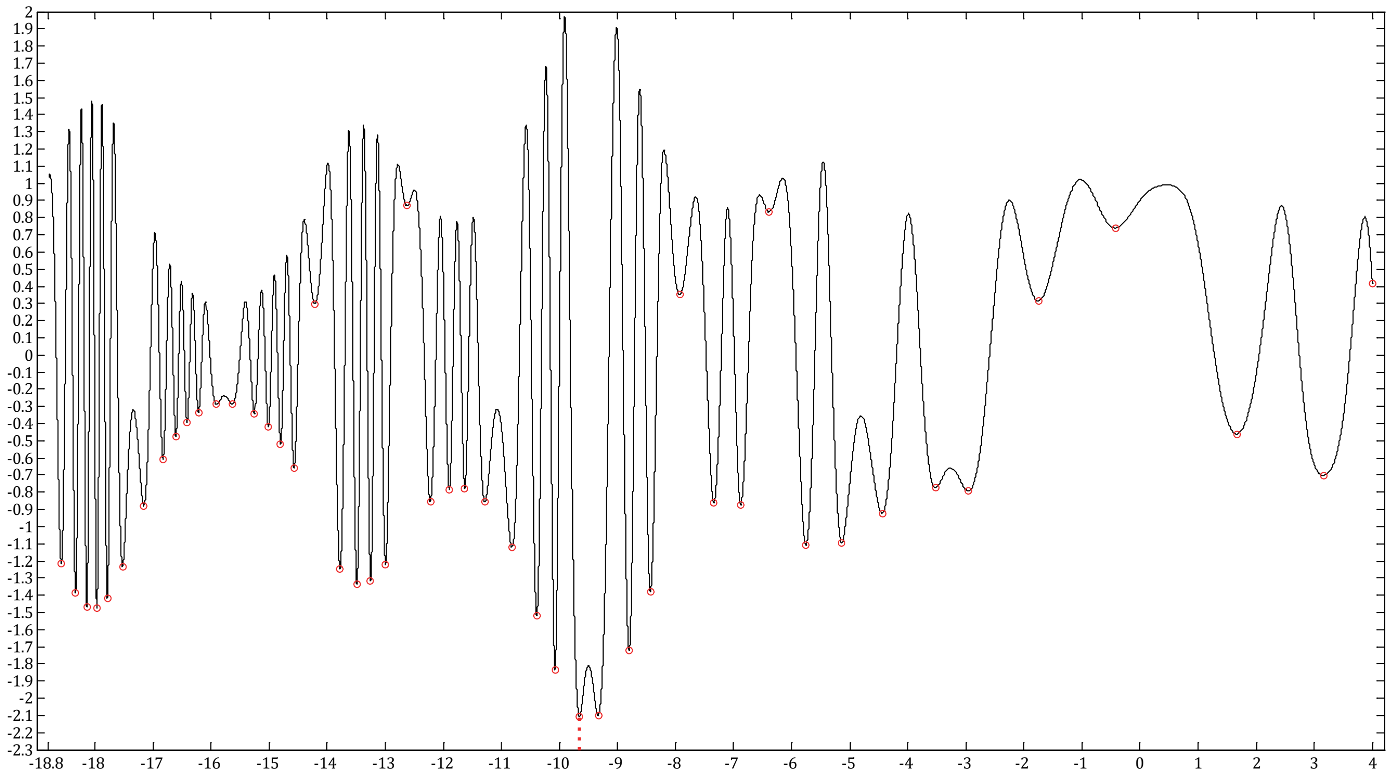

Fig. 7. The 48 local minima, including only the local right boundary minima and the global minimum (marked by the dotted line downward) of function (44) on interval [−18.8; 4].

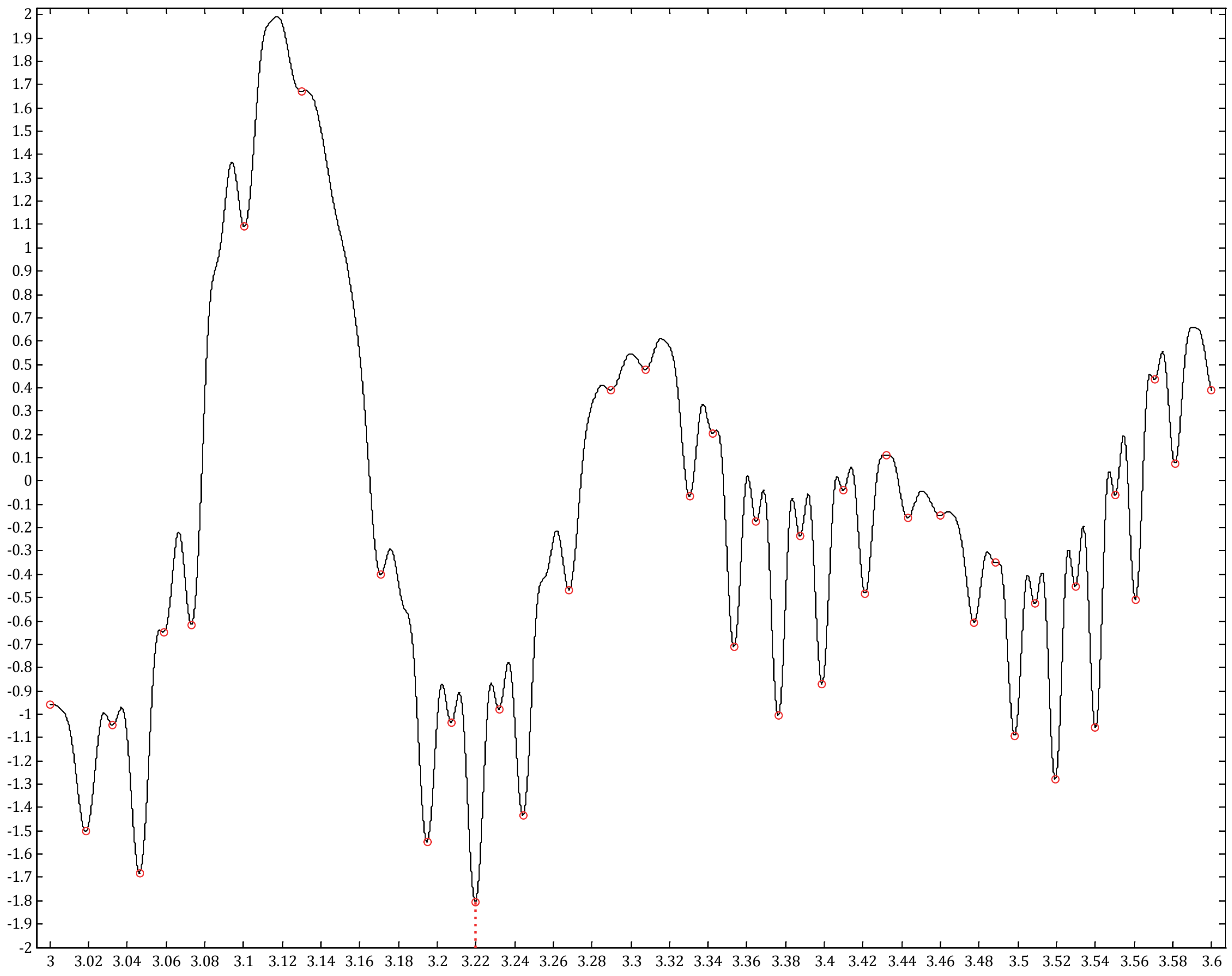

Fig. 8. The 41 local minima, including both the local boundary minima and the global minimum (marked by the dotted line downward) of function (45) on interval [3; 3.6].

VIII. Discussion

The presented approach is applicable to whichever task of finding local extrema is. It is clear that if primarily the purpose is to find all local maxima or the global maximum of the function, the presented approach is applied to function $-f(x)$.

If the function is unimodal, the three-point iterated interval half-cutting is slower than the golden-section search. The slowdown is relatively significant if to measure the computational time for a long series of the minimization problems. Nevertheless, the function unimodality is a rare occasion in real-world practice. There are a few local extrema at least. Even if not all local minima (or maxima) are to be found, the golden-section search may fail to "hit" the global minimum (or maximum), whereas the three-point iterated interval half-cutting completes the task.

Without knowing additional information about the function, unfortunately, there cannot be assurance of that every local minimum will be included into the output set X_*. Doubling the number of subintervals to see whether the same minima remain, when (35) and (37) are expected to be true, may fail in a case if the function (strictly speaking, the data) is overly fluctuating (something similar to Figs. 7 and 8). Then set X_* lacks some minima. If a researcher deals with presumably highly fluctuating functions (data), the part in algorithm #3 can be slightly modified: if (34) holds then algorithm #2 can be applied by the number of subintervals mN_j, where $m > 2$. Reversely, studying rare-fluctuating functions (data) may be more efficient if, for instance, $m = 1.5$ or about that.

The presented approach is a significant and important contribution to the field of numerical estimation and approximate analysis. In real-world contemporary practice, it must serve as a computationally efficient tool for fine-tuning neural networks, adjusting parameters of complex systems (like radars, radio telescopes, massive engines, big-scaled constructions, etc.), and optimal parametrization at all.

IX. Conclusion

The suggested method of three-point iterated interval half-cutting is an approach to finding all local extrema of an unknown single-variable function bounded on a given interval regardless of the interval length. The three of six inputs of the method routine are straightforwardly defined, whereas the tolerance with the minimal and maximal numbers of subintervals are adjustable. These subinterval numbers are primary adjustable inputs. Although the method does not assure obtaining all local minima (or maxima) for any function, setting appropriate minimal and maximal numbers of subintervals makes missing some minima (or maxima) very unlikely. The tolerance is the secondary adjustable input. If the minimal and maximal numbers of subintervals are selected too small, setting whichever small tolerance cannot help in finding every local extremum.

The endpoints of the initial interval and a formula for evaluating the single-variable function at any point of this interval are inputted along with the three adjustable inputs. Having broken the initial interval into a set of subintervals, the three-point iterated interval half-cutting "gropes" around every local minimum by successively cutting off a half of the subinterval or dividing the subinterval in two. A range of subinterval sets defined by the minimal and maximal numbers of subintervals is covered by running the three-point interval half-cutting on every set of subintervals. As a set of values of currently found local minima points changes no more than by the tolerance, the set of local minimum points and the respective set of minimum values of the function are returned.

References

[1] M. L. Lial, R. N. Greenwell, and N. P. Ritchey, *Calculus with Applications (11th edition)*. Pearson, 2016.

[2] I. Zelinka, V. Snášel, A. Abraham, Eds. *Handbook of Optimization. From Classical to Modern Approach*. Springer-Verlag Berlin Heidelberg, 2013. https://doi.org/10.1007/978-3-642-30504-7

[3] S. A. Vavasis, "Complexity issues in global optimization: A survey," in *Handbook of Global Optimization. Nonconvex Optimization and Its Applications,* vol. 2, R. Horst and P. M. Pardalos, Eds. Springer, Boston, MA, 1995, pp. 27–41. https://doi.org/10.1007/978-1-4615-2025-2_2

[4] J. Stewart, *Calculus: Early Transcendentals (6th edition)*. Brooks/Cole, 2008.

[5] E. Hewitt and K. R. Stromberg, *Real and Abstract Analysis*. Springer, 1965. https://doi.org/10.1007/978-3-642-88044-5

[6] K. R. Stromberg, *Introduction to Classical Real Analysis*. Wadsworth, 1981.

[7] L. D. Hoffmann, G. L. Bradley, and K. H. Rosen, *Applied Calculus for Business, Economics, and the Social and Life Sciences*. McGraw-Hill Higher Education, 2005.

[8] R. Fletcher, *Practical Methods of Optimization (2nd edition)*. J. Wiley and Sons, Chichester, 1987.

[9] V. V. Romanuke, "Appropriate number and allocation of ReLUs in convolutional neural networks," *Research Bulletin of NTUU "Kyiv Polytechnic Institute"*, no. 1, pp. 69–78, Mar. 2017. https://doi.org/10.20535/1810-0546.2017.1.88156

[10] V. V. Romanuke, "Appropriateness of DropOut layers and allocation of their 0.5 rates across convolutional neural networks for CIFAR-10, EEACL26, and NORB datasets," *Applied Computer Systems*, vol. 22, no. 1, pp. 54–63, Dec. 2017. https://doi.org/10.1515/acss-2017-0018

[11] V. V. Romanuke, "Wind farm energy and costs optimization algorithm under uncertain parameters of wind speed distribution," *Studies in Informatics and Control*, vol. 27, no. 2, pp. 155–164, 2018. https://doi.org/10.24846/v27i2y201803

[12] V. V. Romanuke, "Time series smoothing and downsampling for improving forecasting accuracy," *Applied Computer Systems*, vol. 26, no. 1, pp. 60–70, May 2021. https://doi.org/10.2478/acss-2021-0008

[13] V. V. Romanuke, "Multiple direction interference suppression by uniform linear phased array sidelobe efficient canceller," *Information and Telecommunication Sciences*, vol. 12, no. 1, pp. 33–40, Jun. 2021. https://doi.org/10.20535/2411-2976.12021.33-40

[14] P. von Butovitsch (ed.), *Advanced Antenna Systems for 5G Network Deployments: Bridging the Gap Between Theory and Practice*. Cambridge, Massachusetts, USA, Academic Press, 2020. https://doi.org/10.1016/C2018-0-05274-3

[15] J. Kiefer, "Sequential minimax search for a maximum," *Proceedings of the American Mathematical Society*, vol. 4, no. 3, pp. 502–506, 1953. https://doi.org/10.2307/2032161

[16] M. Avriel and D. J. Wilde, "Optimality proof for the symmetric Fibonacci search technique," *Fibonacci Quarterly*, no. 4, pp. 265–269, 1966.

[17] W. H. Press, "Minimization or maximization of functions," in *Numerical Recipes: The Art of Scientific Computing (3rd edition)*, W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Eds. Cambridge University Press, New York, 2007, pp. 487–562.

[18] A. Kheldoun, R. Bradai, R. Boukenoui, and A. Mellit, "A new Golden Section method-based maximum power point tracking algorithm for photovoltaic systems," *Energy Conversion and Management*, no. 111, pp. 125–136, Mar. 2016. https://doi.org/10.1016/j.enconman.2015.12.039

[19] K. J. Overholt, "Efficiency of the Fibonacci search method," *BIT Numerical Mathematics*, vol. 13, no. 1, pp. 92–96, Mar. 1973. https://doi.org/10.1007/BF01933527

[20] J.-D. Lee, C.-H. Chen, J.-Y. Lee, L.-M. Chien, and Y.-Y. Sun, "The Fibonacci search for cornerpoint detection of two-dimensional images," *Mathematical and Computer Modelling: An International Journal*, vol. 16, no. 11, pp. 15–20, Nov. 1992. https://doi.org/10.1016/0895-7177(92)90102-Q

[21] S. Edelkamp and S. Schrödl, "Chapter 7 – Symbolic search," in *Heuristic Search*, S. Edelkamp, S. Schrödl, Eds. Morgan Kaufmann, 2012, pp. 283–318. https://doi.org/10.1016/B978-0-12-372512-7.00007-9

[22] Ş. E. Amrahov, A. S. Mohammed, and F. V. Çelebi, "New and improved search algorithms and precise analysis of their average-case complexity," *Future Generation Computer Systems*, vol. 95, pp. 743–753, Jun. 2019. https://doi.org/10.1016/j.future.2019.01.043

[23] L. D. Chambers, Ed. *The Practical Handbook of Genetic Algorithms: Applications (2nd edition)*. Chapman and Hall/CRC, 2000. https://doi.org/10.1201/9781420035568

[24] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

Vadim V. Romanuke was born in 1979. He graduated from the Technological University of Podillya in 2001. Higher education was obtained in 2001. In 2006, he received the Degree of Candidate of Technical Sciences in Mathematical Modelling and Computational Methods. The degree of Doctor of Technical Sciences in Mathematical Modelling and Computational Methods was received in 2014. In 2016, Vadim Romanuke received the academic status of Full Professor. His current research interests concern wireless communication systems, job scheduling, semantic image segmentation, decision making, game theory, statistical approximation, and control engineering based on statistical correspondence.
Address for correspondence: Soborna str., 87, Vinnytsia, Ukraine, 21050.
E-mail: romanukevadimv@gmail.com
ORCID iD: https://orcid.org/0000-0001-9638-9572