Improvements of the Modified Anderson-Björck (modAB) Root-Finding Algorithm

Nedelcho Ganchovski * , Oscar Smith , Christopher Rackauckas , Lachezar Tomov , Alexander Traykov

Posted Date: 27 March 2026

doi: 10.20944/preprints202603.2190.v1

Keywords: root-finding algorithm; bracketing method; nonlinear equation solver

 doi:10.20944/preprints202603.2190.v1

Article

Improvements of the Modified Anderson-Björck (modAB) Root-Finding Algorithm

Nedelcho Ganchovski [1,*], Oscar Smith [2], Christopher Rackauckas [2,3], Lachezar Tomov [4] and Alexander Traykov [5]

[1] Proektsoft EOOD, Sofia, Bulgaria
[2] JuliaHub, Cambridge, Massachusetts, USA
[3] Massachusetts Institute of Technology, Cambridge, Massachusetts, USA
[4] New Bulgarian University, Sofia, Bulgaria
[5] University of Architecture, Civil Engineering and Geodesy, Sofia, Bulgaria
* Correspondence: proektsoft.bg@gmail.com

Abstract

Modified Anderson-Björck's method [1] is a new robust and efficient bracketing root finding algorithm. It combines bisection with Anderson-Björk's method to achieve both fast performance and worst-case optimality. It relies on linearity check criteria for switching methods and uses Anderson-Björk corrections to overcome the fixed endpoint issue of false-position. Initial benchmarks of this method have shown certain performance advantages compared to other methods like Ridders, Brent and ITP. In this paper, we propose further improvements of the method and perform some additional analysis and benchmarks of its behavior and performance.

Keywords: root-finding algorithm; bracketing method; nonlinear equation solver

1. Introduction

Iterative algorithms are generally fast but can easily diverge, if improper starting point is selected. Bracketing algorithms are more reliable in this respect. If the function is continuous in the enclosing interval [a, b] and has opposite signs at both ends, according to the Bolzano's theorem, at least one root exists inside [a, b] and the bracketing method is guaranteed to find it with a certain precision. Bisection is the simplest method, and also worst-case optimal. The required number of iterations is constant for a certain precision and does not depend on the function properties. It can be precomputed by the following equation:

$$n = \log_2 \frac{|x_2 - x_1|}{2\epsilon} \tag{1}$$

However, for well-behaved functions, other algorithms like false position, Illinois [6] and Ridder's [10] can provide significantly better performance, but for poor-behaved ones they can downgrade much below bisection. So, stability and performance are contradictory goals and achieving both is a challenging task. Some authors have noticed that this can be done by combining bisection with a faster algorithm, like false-position [13], secant or inverse-quadratic interpolation. Such are the Brent's [11] and ITP [9] methods. Also, initial bisection steps can boost the performance of false-position and similar methods by bringing the starting point to more favorable conditions [1,5]. Although obtained by combining only linear algorithms, the resulting algorithm shows nonlinear (bilinear) behavior and better convergence than using both separately. In all cases, the selection of proper switching criteria is of key importance for applying each method at the most appropriate place and thus maximizing the overall performance.

 doi:10.20944/preprints202603.2190.v1

2. The Original Modified Anderson-Björck (modAB) Algorithm

On that basis, a new hybrid algorithm was proposed in 2022 [1] that combined bisection with Anderson-Björck's (AB) method. The switching criteria is based on measuring the linearity of the function. In this way, linear interpolation is applied when the deviation of the curve from a straight line becomes small enough according to the equation:

$$|y_3 - y_m| < k(|y_3| + |y_m|), \tag{2}$$

where y_3 is the function value at midpoint, y_m is the chord ordinate at the same point and k is the limit deviation factor. After switching, the iterations continue as ordinary AB steps as follows:

1. An intermediate point is calculated by linear interpolation:

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)} \tag{3}$$

2. The ordinate of the end that remained fixed at the previous iteration is reduced by a factor m. For the left end, it is calculated by the expression: $y_1 = f(x_1) \cdot m$, where $m = 1 - f(x_3) / f(x_2)$ if $m > 0$, otherwise: $m = 0.5$. For the right end, the equation is obtained by swapping indices 1 and 2, as follows: $y_2 = f(x_2) \cdot m$, where $m = 1 - f(x_3) / f(x_1)$ if $m > 0$, otherwise: $m = 0.5$

An option to fall back to bisection is provided, if AB fails to converge within a certain number of iterations. The algorithm was included in Calcpad [2] and Root-Fortan [3] libraries. Although the proposed algorithm showed superior performance compared to others in the test suite and later in practice, we identified some drawbacks and opportunities for further improvements.

3. Drawbacks of the Original modAB Method

The previously proposed algorithm has several shortcomings that can be addressed:

3. The deviation factor k in eq. (2) has a fixed value of $k = 0.25$ as a good average estimate that is experimentally tuned on a larger set of functions. However, this is not optimal because it does not account for the properties of each function in the selected interval.
4. Although the possibility of redundant switch to AB was discussed in [1], nothing much has been done on the subject. Only a simple bisection fallback is included as a safety mechanism, if the algorithm fails to converge within the expected number of iterations. It is proposed empirically to be half of the required bisection steps. This is a simple and conservative approach, but in some cases, it may result in a "waste" of AB steps before it "realizes" that it is not working as required.
5. The original algorithm uses the termination criteria of the false-position method: $|x_n - x_{n-1}| < \varepsilon$. Further tests have shown that it fails for certain functions that are flat around the root, but with large ordinates at endpoints ($> 1/\varepsilon$). This causes the interpolation point to crawl very slowly along the abscissa, so the step can become lower than ε even if we are still too far from the root.
6. The benchmarks in the original paper [1] were based on the number of iterations with a factor of two for Ridder's method. However, the number of function calls is more closely related to the actual execution speed when the algorithm is implemented in software.
7. In the original paper, we used 21 functions to benchmark the method performance. This set is very limited, which can bring some distortions to the results. So, one may consider it insufficient to derive reliable conclusions. It may occur that some algorithms are particularly suitable for a larger part of the functions in the set showing a false advantage. That is why additional tests were performed in [5] with a set of 50 functions.

4. Improvements of the modAB Algorithm

In this paper, we propose a new version of the modAB algorithm aiming to be more robust, reliable and efficient than the first one. This is achieved by fixing the above issues and making it more adaptive to the properties of the function.

 doi:10.20944/preprints202603.2190.v1

4.1. Adaptive Value of Deviation Factor k

For highly curved functions, like the one on Figure 1, symmetry plays a great role in the behavior of linear interpolation algorithms. If endpoint ordinates are highly asymmetric like on the left plot, there is a possibility for the root to occur at a greater distance. This will slow down performance since there is not only a longer path to travel, but the step size is also smaller. If the endpoint values are nearly symmetric, the behavior improves significantly, although both cases have the same deviation from the straight line.

$$11 \cdot x^{11} - 1 = 0 \ @ \ x \in [0.1; 1] \qquad 11 \cdot x^{11} - 5 = 0 \ @ \ x \in [0.1; 1]$$

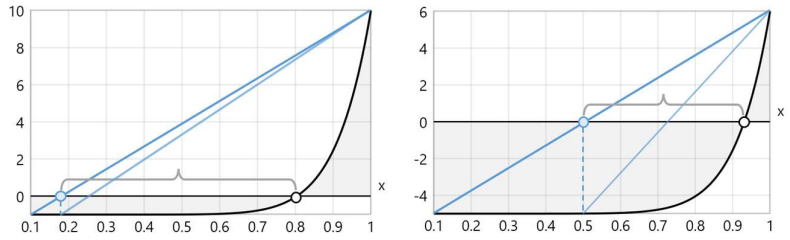

Figure 1. Plots of function $f(x) = 11 \cdot x^{11} - c$ for $c = 1$ and 5.

That is why it is reasonable to include symmetry into the method switching criterion and the factor k seems to be the right place. For unsymmetric endpoints, we can keep k lower, forcing the algorithm to make more bisection steps. For symmetric endpoints we can keep k larger and allow the algorithm to switch faster to AB since we expect it to be more favorable. However, this is highly dependent on the shape of the function. So, there is no sense in tuning it too much for specific types of functions, because it may not work for other types. Instead, we can just map the value of k against the endpoint ordinates by using a simple relationship that is fast to evaluate and does not bring significant overhead on each iteration.

For that purpose, we will introduce a new factor of symmetry r by using the equation:

$$r = 1 - \left| \frac{y_m}{y_2 - y_1} \right|, \tag{4}$$

where $y_1 = f(x_1)$, $y_2 = f(x_2)$ and $y_m = 0.5\,(y_1 + y_2)$ is the average value. Thus, we have $r = 1$ for symmetric and $r = 0.5$ for unsymmetric endpoint ordinates. Symmetry also ensures that the distance from the interpolation point to the root cannot exceed $0.5\,(x_2 - x_1)$. Otherwise, it can approach the whole interval length of $x_2 - x_1$. But how to account for it in the deviation factor k? If we look closely at the above functions, we can notice that the relationship between symmetry and convergence is not linear. Higher symmetry leads to both shorter path and longer step, which is rather quadratic than linear. The simplest quadratic expression we can use is $k = r^2$. Thus, we map k to the interval [0.25, 1], where 0.25 is our old value for k, but with allowance to grow up to 1 for fully symmetric endpoints.

4.2. Adaptive Bisection Fallback Criterion

In this paper, we propose a new version of the modAB algorithm aiming to be more robust, reliable and efficient than the first one. This is achieved by fixing the above issues and making it more adaptive to the properties of the function. Instead of using a fixed number of expected iterations, we can apply more precise criterion, by measuring the actual performance of AB algorithm after switching. One very simple and direct approach is to check whether AB has shrunk the interval more than bisection would do. Assume that the algorithm switches from bisection to AB at the interval $[a_0, b_0]$ with initial width:

$$W_0 = b_0 - a_0 \tag{5}$$

After performing t more steps, a pure bisection method would reduce the width to $W_0/2^t$. So, we will define a fallback threshold as follows:

 doi:10.20944/preprints202603.2190.v1

$$T(t) = C\frac{W_0}{2^t} \tag{6}$$

where C is a safety factor. Let $W(t) = b_t - a_t$ denotes the actual interval width after t AB steps. Then, the adaptive bisection fallback criterion becomes:

$$W(t) > T(t) \tag{7}$$

Taking $C = 2^n$ will allow for a delay of n AB steps before switching back to bisection. Numerical experiments have shown that without a safety factor for certain functions the algorithm starts to oscillate rapidly between both methods, causing more "damage" than improvement. The question is how to select a suitable value for C. One or two iterations proved to be insufficient to prevent oscillations. A reasonable value of n appeared to be about 3 to 4. Larger values caused the algorithm to waste too many iterations on AB. So, we can safely pick $n = 4$ and thence $C = 2^4 = 16$. For efficiency, we will not apply equation (6) directly but instead will initialize the threshold at the moment of switching as $T_0 = CW_0$. Then we will update it on each AB iteration t as follows: $T_t = T_{t-1}/2$.

To demonstrate how the new method switching criteria work, we will take a nearly symmetrical function $f(x) = x^3 - 0.001$ and solve the equation $f(x) = 0$ in the interval [-10, 10]. The plot is displayed on Figure 2.

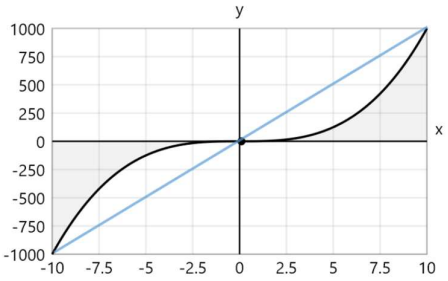

Figure 2. Plot of function $f(x) = x^3 - 0.001$ for $x \in [-10, 10]$.

Both original and proposed algorithms switch to AB at the first iteration, finding itself in a position to deal with a highly nonlinear and unsymmetric problem. But after that, they show very different behavior as summarized in the following table:

This demonstrates that using fixed switching criteria is not optimal and can possibly slow down the algorithm.

Table 1. Comparison between the original and the proposed method switching criteria.

Algorithm	Iteration number when			
	switch to AB	fallback to bisection	switch to AB	converge and quit
The original modAB 2022 fixed criteria	1	24	34	40
The proposed new adaptive criteria	1	7	15	24

4.3. New Termination Check

Instead of the false-position convergence check $|x_n - x_{n-1}| < \varepsilon$, we will use the more robust and reliable bisection check $b_n - a_n < \varepsilon_{abs} + \varepsilon_{rel} \cdot |x_n|$, where a_n and b_n are the endpoints of the interval at iteration n, and ε_{abs} and ε_{rel} are the absolute and relative tolerances, respectively. This criterion is not applicable for the pure false-position method because of the fixed endpoint issue. But having AB corrections, the interval will eventually shrink from both sides. However, for some functions this may require 1 or 2 extra steps after the solution has actually converged, just to shrink the interval from the other side. But it is a good trade to provide robustness and reliability of the method.

 doi:10.20944/preprints202603.2190.v1

4.4. The Improved Algorithm

After implementing the above improvements and refactoring the structure, we arrive at the following improved version of the modAB algorithm:

ALGORITHM 1: The improved modAB algorithm

Calculate points p_1: $y_1 = f(x_1)$ and p_2: $y2 = f(x_2)$
initialize "method" to "bisection" and "side" to "none"
for iter = 1 to *maxIterations*
  if "method" is "bisection" then:
    calculate p_3 as the midpoint such that $x_3 = (x_1 + x_2) / 2$ and $y_3 = f(x_3)$
    calculate the average ordinate by the equation: $y_m = (y_1 + y_2) / 2$
    calculate r to equation (4) and $k = r^2$
      if the function is close enough to linear, satisfying that
      $|y_m - y_3| < k\,(|y_3| + |y_m|)$ then:
      change the method to "Anderson-Björck"
    end
  else if method is "Anderson-Björck" then:
    calculate p_3 as the secant point such that
    $x_3 = (x_1 * y_2 - y_1 * x_2) / (y_2 - y_1)$ and $y_3 = f(x_3)$
    clamp p_3 between p_1 and p_2 in case floating point
    rounding errors shoots it outside the interval
  end
  if the convergence check:
$y_3 = 0$ or $x_2 - x_1 \le \varepsilon\_{abs} + \varepsilon\_{rel} * |x_3|$ is satisfied then:
  stop and return x_3 as a result
  if y_1 and y_3 have equal signs then:
    if "method" is "Anderson-Björck" then: (*apply AB correction*)
      if "side" is "right" then
        calculate $m = 1 - y_3 / y_1$
        if $m \le 0$ then: divide y_2 by 2 else: multiply y_2 by m
      else:
        change "side" to "right"
      end
    end
    for both methods, shrink the interval from left by setting $p_1 = p_3$
  else
    if "method" is "Anderson-Björck" then:
      if "side" is "left" then:
        calculate $m = 1 - y_3 / y_2$
        if $m \le 0$ then: divide $y1$ by 2 else: multiply $y1$ by m
      else:
        change "side" to "left"
      end
    end
    for both methods, shrink the interval from right by setting $p_2 = p_3$
  end
  if AB fails to shrink the interval enough, then:
    reset the method back to "bisection"
end
return error, if failed to converge for the expected number of iterations

 doi:10.20944/preprints202603.2190.v1

The source code of the algorithm is listed in Appendix A. It was also translated into Julia, C++ and Python/Cython languages and included in Calcpad [2], SciML/NonlinearSolve.jl [4], JuliaMath/Roots.jl [21], and ROOT.CERN [22] libraries for scientific computations.

5. Numerical Experiments

The improved modAB algorithm was benchmarked against the original version and seven classical algorithms: bisection, false-position, Illinois [6], Anderson-Björck [8], ITP [9], Ridders [10] and Brent [11,20]. For that purpose, they were implemented in C# as well.

An extended set of 92 functions is used for benchmarking they were collected almost entirely from papers from the authors of other algorithms and independent benchmarks, as follows:

- Sérgio Galdino [15]: $f_{01} - f_{30}$
- Custom functions [5]: $f_{31} - f_{32}$
- Steven A. Stage [16]: $f_{33} - f_{40}$
- Swift & Lindfield [17]: $f_{41} - f_{50}$
- Alojz Suhadolnik [18]: $f_{51} - f_{62}$
- Oliveira & Takahashi [9]: $f_{63} - f_{83}$
- SciML Benchmarks suite [4]: $f_{84} - f_{92}$

The benchmarks are based on the number of function evaluations, which are more indicative for the actual performance than the number of iterations, especially for computationally heavy functions. The results are presented in the following table:

Table 2. Test functions and benchmark results for solving $f_{nn}(x) = 0$ on the interval [a; b].

Function	Interval		Method								
	a	*b*	bs	fp	ill	AB	ITP	Rid	Bre	modAB¹	modAB²
$f_{01}(x) = x^3 - 1$	0.5	1.5	**3**	35	11	9	10	4	11	**3**	**3**
$f_{02}(x) = x^2(x^2/3 + \text{sqrt}(2)\cdot\sin(x)) - \text{sqrt}(x)/18$	0.1	1	49	88	13	13	39	16	14	**12**	13
$f_{03}(x) = 11x^{11} - 1$	0.1	1	49	109	17	24	50	14	**13**	**13**	16
$f_{04}(x) = x^3 + 1$	-1.8	0	50	51	13	12	12	12	11	**10**	**10**
$f_{05}(x) = x^3 - 2x - 5$	2	3	48	33	10	**9**	48	14	**9**	10	13
$f_{06}(x) = 2x\cdot\exp(-5) + 1 - 2\cdot\exp(-5x)$	0	1	50	31	**10**	11	13	14	11	12	**10**
$f_{07}(x) = 2x\cdot\exp(-10) + 1 - 2\cdot\exp(-10x)$	0	1	50	32	13	12	15	14	13	13	**11**
$f_{08}(x) = 2x\cdot\exp(-20) + 1 - 2\cdot\exp(-20x)$	0	1	50	32	14	14	17	**12**	13	13	**12**
$f_{09}(x) = (1 + (1-5)^2)\cdot x^2 - (1-5x)^2$	0	1	50	18	13	**10**	48	16	11	12	12
$f_{10}(x) = (1 + (1-10)^2)\cdot x^2 - (1-10x)^2$	0	1	50	15	11	**9**	12	16	11	**9**	**9**
$f_{11}(x) = (1 + (1-20)^2)\cdot x^2 - (1-20x)^2$	0	1	50	13	11	**9**	10	16	10	**9**	**9**
$f_{12}(x) = x^2 - (1-x)^5$	0	1	50	42	11	**10**	11	14	**10**	12	12
$f_{13}(x) = x^2 - (1-x)^{10}$	0	1	50	84	13	**11**	49	16	**11**	12	13
$f_{14}(x) = x^2 - (1-x)^{20}$	0	1	50	172	13	13	50	16	14	13	**12**
$f_{15}(x) = (1 + (1-5)^4)\cdot x - (1-5x)^4$	0	1	50	10	10	**8**	10	12	9	9	9
$f_{16}(x) = (1 + (1-10)^4)\cdot x - (1-10x)^4$	0	1	50	**7**	8	**7**	9	12	8	8	9
$f_{17}(x) = (1 + (1-20)^4)\cdot x - (1-20x)^4$	0	1	50	**7**	8	**7**	10	12	8	8	9
$f_{18}(x) = \exp(-5x)\cdot(x-1) + x^5$	0	1	49	82	11	**9**	37	14	10	11	12
$f_{19}(x) = \exp(-10x)\cdot(x-1) + x^{10}$	0	1	49	202	15	**10**	11	16	**10**	13	13
$f_{20}(x) = \exp(-20x)\cdot(x-1) + x^{20}$	0	1	49	202	23	**11**	12	16	14	13	15
$f_{21}(x) = x^2 + \sin(x/5) - 1/4$	0	1	50	35	11	10	10	12	12	12	**9**
$f_{22}(x) = x^2 + \sin(x/10) - 1/4$	0	1	49	34	12	10	10	12	11	**9**	**9**
$f_{23}(x) = x^2 + \sin(x/20) - 1/4$	0	1	49	33	13	10	10	12	11	**9**	**9**
$f_{24}(x) = (x+2)\cdot(x+1)\cdot(x-3)^3$	2.6	4.6	**49**	202	92	109	**49**	80	138	**49**	**49**
$f_{25}(x) = (x-4)^5\cdot\log(x)$	3.6	5.6	**49**	202	175	189	**49**	62	119	**49**	**49**
$f_{26}(x) = (\sin(x) - x/4)^3$	2	4	**49**	202	92	109	**49**	78	138	**49**	**49**

 doi:10.20944/preprints202603.2190.v1

$f_{27}(x) = (81 - p(x) \cdot (108 - p(x) \cdot (54 - p(x) \cdot (12 - p(x))))) \cdot \text{sign}(p(x) - 3)$, where $p(x) = x + 1.11111$	1	3	**14**	202	35	38	22	24	32	**14**	**14**
$f_{28}(x) = \sin(x - 7.143)^3)$	7	8	**47**	202	88	103	**47**	70	132	**47**	**47**
$f_{29}(x) = \exp((x - 3)^5) - 1$	2.6	4.6	12	202	59	46	**3**	18	31	12	12
$f_{30}(x) = \exp((x - 3)^5) - \exp(x - 1)$	4	5	48	202	53	58	48	16	15	**14**	**14**
$f_{31}(x) = \pi - 1/x$	0.05	5	52	193	15	**7**	53	16	13	13	12
$f_{32}(x) = 4 - \tan(x)$	0	1.5	49	93	15	**12**	50	16	14	13	15
$f_{33}(x) = \cos(x) - x^3$	0	4	51	201	16	16	**11**	18	16	13	13
$f_{34}(x) = \cos(x) - x$	-11	9	54	12	11	**9**	11	16	11	10	11
$f_{35}(x) = \text{sqrt}(\text{abs}(x - 2/3)) \cdot \text{if}(x \le 2/3; 1; -1) - 0.1$	-11	9	54	16	17	14	53	24	**12**	18	18
$f_{36}(x) = \text{abs}(x - 2/3)^{0.2} \cdot \text{if}(x \le 2/3; 1; -1)$	-11	9	54	**45**	48	55	46	54	**45**	54	56
$f_{37}(x) = (x - 7/9)^3 + (x - 7/9) \cdot 10^{-3}$	-11	9	53	202	25	27	**13**	24	32	18	16
$f_{38}(x) = \text{if}(x \le 1/3; -0.5; 0.5)$	-11	9	54	54	62	62	**53**	56	54	54	54
$f_{39}(x) = \text{if}(x \le 1/3; -10^{-3}; 1 - 10^{-3})$	-11	9	**54**	202	173	173	**54**	84	84	**54**	**54**
$f_{40}(x) = \text{if}(x = 0; 0.0; 1/(x - 2/3))$	-11	9	54	**9**	101	101	54	60	56	54	54
$f_{41}(x) = 2x \cdot \exp(-5) - 2 \cdot \exp(-5x) + 1$	0	10	53	34	14	**13**	17	14	14	15	**13**
$f_{42}(x) = (x^2 - x - 6) \cdot (x^2 - 3x + 2)$	0	π	51	24	13	**10**	16	18	11	13	13
$f_{43}(x) = x^3$	-1	1.5	**51**	202	95	114	52	84	125	**51**	**51**
$f_{44}(x) = x^5$	-1	1.5	**51**	202	184	202	52	56	128	**51**	**51**
$f_{45}(x) = x^7$	-1	1.5	**51**	202	202	202	52	52	136	**51**	**51**
$f_{46}(x) = (\exp(-5x) - x - 0.5)/x^5$	0.09	0.7	49	202	25	23	50	**14**	16	**14**	**14**
$f_{47}(x) = 1/\text{sqrt}(x) - 2\log(5 \cdot 10^3 \cdot \text{sqrt}(x)) + 0.8$	0.0005	0.5	49	156	17	**14**	49	18	17	16	17
$f_{48}(x) = 1/\text{sqrt}(x) - 2\log(5 \cdot 10^7 \cdot \text{sqrt}(x)) + 0.8$	0.0005	0.5	49	45	16	**14**	20	18	17	19	15
$f_{49}(x) = \text{if}(x \le 0; -x^3 - x - 1; x^{1/3} - x - 1)$	-1	1	50	27	13	**10**	12	12	12	12	13
$f_{50}(x) = x^3 - 2x - x + 3$	-3	2	51	42	15	13	16	14	13	12	**11**
$f_{51}(x) = \log(x)$	0.5	5	51	29	12	**10**	12	12	11	11	**10**
$f_{52}(x) = (10 - x) \cdot \exp(-10x) - x^{10} + 1$	0.5	8	52	202	39	21	53	18	18	**15**	16
$f_{53}(x) = \exp(\sin(x)) - x - 1$	1.0	4	50	33	**12**	**12**	**12**	18	15	13	14
$f_{54}(x) = 2\sin(x) - 1$	0.1	$\pi/3$	49	17	9	**8**	12	12	9	10	11
$f_{55}(x) = (x - 1) \cdot \exp(-x)$	0.0	1.5	50	71	13	10	12	**4**	12	10	10
$f_{56}(x) = (x - 1)^3 - 1$	1.5	3	49	58	13	**10**	11	14	12	11	11
$f_{57}(x) = \exp(x^2 + 7x - 30) - 1$	2.6	3.5	48	202	21	31	48	14	13	12	**11**
$f_{58}(x) = \text{atan}(x) - 1$	1.0	8	51	27	12	**9**	13	12	11	12	10
$f_{59}(x) = \exp(x) - 2x - 1$	0.2	3	50	148	16	15	**12**	14	14	**12**	13
$f_{60}(x) = \exp(-x) - x - \sin(x)$	0.0	2	51	16	10	**8**	11	16	9	9	9
$f_{61}(x) = x^2 - \sin(x)^2 - 1$	-1	2	50	34	14	14	13	18	**12**	**12**	14
$f_{62}(x) = \sin(x) - x/2$	$\pi/2$	π	49	33	11	**9**	10	14	10	10	11
$f_{63}(x) = x \cdot \exp(x) - 1$	-1	1	50	31	12	**10**	12	10	11	11	11
$f_{64}(x) = \tan(x - 1/10)$	-1	1	51	41	9	**8**	11	12	10	9	**8**
$f_{65}(x) = \sin(x) + 0.5$	-1	1	50	12	**9**	**9**	11	16	**9**	**9**	**9**
$f_{66}(x) = 4x^5 + x^2 + 1$	-1	1	50	34	14	12	15	16	**11**	13	14
$f_{67}(x) = x + x^{10} - 1$	-1	1	50	51	13	**11**	50	12	13	14	12
$f_{68}(x) = \pi^x - e$	-1	1	50	16	10	**8**	11	10	9	9	9
$f_{69}(x) = \log(\text{abs}(x - 10/9))$	-1	1	51	65	13	**10**	**10**	12	11	**10**	**10**
$f_{70}(x) = 1/3 + \text{sign}(x) \cdot \text{abs}(x)^{1/3} + x^3$	-1	1	51	14	14	**12**	16	16	**12**	**12**	18
$f_{71}(x) = (x + 2/3)/(x + 101/100)$	-1	1	50	202	17	**7**	50	16	14	10	8
$f_{72}(x) = (x \cdot 10^6 - 1)^3$	-1	1	51	**4**	**4**	**4**	51	6	140	51	51
$f_{73}(x) = \exp(x) \cdot (x \cdot 10^6 - 1)^3$	-1	1	51	202	96	115	51	**6**	142	51	51
$f_{74}(x) = (x - 1/3)^2 \cdot \text{atan}(x - 1/3)$	-1	1	**51**	202	94	113	**51**	82	143	**51**	**51**

 doi:10.20944/preprints202603.2190.v1

$f_{75}(x) = \text{sign}(3x-1)\cdot(1 - \text{sqrt}(1-(3x-1)^2/81))$	-1	1	27	202	28	37	**23**	38	67	27	27
$f_{76}(x) = \text{if}(x > (1-10^6)/10^6; (1+10^6)/10^6; -1)$	-1	1	50	50	**38**	**38**	49	52	50	50	40
$f_{77}(x) = \text{if}(x \neq 1/21; 1/(21x-1); 0)$	-1	1	51	**15**	96	96	51	66	54	51	51
$f_{78}(x) = x^2/4 + \text{ceiling}(x/2) - 0.5$	-1	1	51	51	57	56	50	40	52	51	**10**
$f_{79}(x) = \text{ceiling}(10x-1) + 0.5$	-1	1	51	49	53	53	47	38	48	**15**	**15**
$f_{80}(x) = x + \sin(x\cdot 10^6)/10 + 10^{-3}$	-1	1	51	25	23	23	27	26	**22**	25	26
$f_{81}(x) = \text{if}(x > -1; 1 + \sin(1/(x+1)); -1)$	-1	1	50	53	16	**15**	50	40	52	20	**15**
$f_{82}(x) = 202x - 2\cdot\text{floor}((2x+10^{-2})/(2\cdot 10^{-2})) - 0.1$	-1	1	51	**4**	**4**	**4**	13	6	5	5	6
$f_{83}(x) = (202x - 2\cdot\text{floor}((2x+10^{-2})/(2\cdot 10^{-2})) - 0.1)^3$	-1	1	**51**	202	74	71	**51**	70	126	**51**	**51**
$f_{84}(x) = (x-1)\cdot(x-2)\cdot(x-3)\cdot(x-4)\cdot(x-5) - 0.05$	0.5	5.5	50	67	10	**8**	10	20	9	**8**	9
$f_{85}(x) = \sin(x) - 0.5x - 0.3$	-10.0	10.0	53	15	14	**12**	47	20	13	**12**	**12**
$f_{86}(x) = \exp(x) - 1 - x - x^2/2 - 0.005$	-2.0	2.0	52	202	20	20	22	20	17	**15**	16
$f_{87}(x) = 1/(x-0.5) - 2 - 0.05$	0.6	2.0	49	137	15	**6**	50	16	13	8	10
$f_{88}(x) = \log(x) - x + 2 - 0.05$	0.1	3.0	51	30	14	**12**	17	16	14	14	15
$f_{89}(x) = \sin(20x) + 0.1x - 0.1$	-4.0	5.0	52	18	18	16	17	20	18	16	**14**
$f_{90}(x) = x^3 - 2x^2 + x - 0.025$	-1.0	2.0	52	176	14	14	52	18	16	**13**	**13**
$f_{91}(x) = x\cdot\sin(1/x) - 0.1 - 0.01$	0.01	1.0	50	14	14	**12**	39	14	14	13	14
$f_{92}(x) = x^3 - 0.001$	-10	10	54	202	**25**	202	55	28	**25**	42	26
Total number of evaluations			4498	8200	2975	3162	2732	2256	2972	1857	1797
Average number of evaluations			48.9	89.1	32.3	34.4	29.7	24.5	32.3	20.2	19.5
Relative to modAB²			2.50	4.56	1.66	1.76	1.52	1.26	1.65	1.03	1.00
Maximal number of evaluations			54	202	202	202	55	84	143	54	56
Geometric mean			47.2	53.8	20.2	18.5	23.4	19.1	19.6	15.9	15.6
Best at (number of functions)			11	7	7	45	15	5	13	31	31
Worst at (number of functions)			40	44	6	7	1	1	1	0	0
Function	*a*	*b*	bs	fp	ill	AB	ITP	Rid	Bre	modAB¹	modAB²

Legend:

bs – Bisection
fp – False-position
ill – Illinois
ITP – Interpolate, truncate, project
Bre – Brent
Rid – Ridders
AB – Anderson-Björck
modAB¹ – Modified Anderson-Björck – the original 2022 version
modAB² – Modified Anderson-Björck – the proposed version

Note: The best algorithm for each function is **highlighted**.

6. Conclusions

The results show about 3% improvement of the new modAB version compared to the old 2022 one, requiring 1797 total evaluations vs 1857. If we keep the same convergence check, the effect from improving the method switching and fallback criteria is about 6% or 1739 total evaluations. So, the new convergence check slows down the algorithm by 3%, but providing robustness is of a greater priority than speed. Compared to the original Anderson-Björck's algorithm, the improvement is about 1.76 times. The new modAB algorithm also outperforms Ridder's by 1.26 times, and ITP and Brent by 1.52 and 1.65 times, respectively. For well-behaved functions, the algorithm either shows the best performance (31 functions) or remains 1-2 evaluations behind the winner. For poorly behaved functions, the guarding role of bisection appears to work, providing no more than 56 evaluations in the worst case. This is probably due to spurious switch from bisection to AB and

 doi:10.20944/preprints202603.2190.v1

fallback. It can be also noted that after extending the number of test functions, the relationships between the efficiencies of the separate algorithms are preserved. This indicates that the previously derived conclusions in [1] were not distorted by the chosen set of functions.

Supplementary Materials: The following supporting information can be downloaded at the website of this paper posted on Preprints.org.

Author Contributions: Conceptualization, Nedelcho Ganchovski; Software, Nedelcho Ganchovski, Oscar Smith and Christopher Rackauckas; Validation, Oscar Smith, Christopher Rackauckas and Lachezar Tomov; Writing – original draft, Nedelcho Ganchovski; Writing – review & editing, Oscar Smith, Lachezar Tomov and Alexander Traykov. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article/supplementary material. Further inquiries can be directed to the corresponding author(s).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Source code of the algorithm in C# programming language.

```
// Struct to store calculation nodes abscissas and ordinates
private struct Node
{
    public double X; public double Y;
    public Node(double x, double y) { X = x; Y = y; }
    public Node(double x, Func<double, double> F) { X = x; Y = F(x); }
    public static double Sec(Node p1, Node p2) =>
        (p1.X * p2.Y - p1.Y * p2.X) / (p2.Y - p1.Y);
    public static double Mid(Node p1, Node p2) => (p1.X + p2.X) / 2.0;
}
// Initializes the interval and checks if bracketing is applicable
private static bool Initialize(double x1, double x2, Func<double, double> F,
out Node p1, out Node p2)
{
    if (x1 > x2) {
        p1 = new(x2, F); p2 = new(x1, F);
    } else {
        p1 = new(x1, F); p2 = new(x2, F);
    }
    EvaluationCount = 0;
    return Math.Sign(p1.Y) != Math.Sign(p2.Y);
}
// Finds the root of F(x) = 0 within the interval [x1, x2] with the specified
// absolute: aTol and relative: rTol precisions, using improved
// modified Anderson Björck's method.
// F(x) must be continuous and sign(F(x1)) ≠ sign(F(x2))
public static double ModAB(Func<double, double> F, double x1, double x2,
double aTol = 1e-14, double rTol = 1e-14, int maxIterations = 200)
{
    if (!Initialize(x1, x2, F, out Node p1, out Node p2))
        return double.NaN;
    var bisection = true; // Initialize the method to bisection
```

 doi:10.20944/preprints202603.2190.v1


```
// Store the side that moved last: -1 for left, 1 for right, 0 for none
var side = 0;
var threshold = p2.X – p1.X; // Threshold to reset back to bisection
const double C = 16; // Safety margin of 4 iterations behind the threshold
for (int i = 1; i <= maxIterations; ++i) {
    Node p3;
    if (bisection) {
        p3 = new Node(Node.Mid(p1, p2), F);
        double y1 = p1.Y, y2 = p2.Y;
        var ym = (y1 + y2) / 2.0;
        var r = 1 - Math.Abs(ym / (y2 - y1)); // Symmetry factor
        var k = r * r; // Deviation factor
        // Check if the function is close enough to straight line
        if (Math.Abs(ym - p3.Y) < k*(Math.Abs(p3.Y) + Math.Abs(ym))) {
            bisection = false;
            // Initialize threshold for bisection fallback
            threshold = (p2.X - p1.X) * C;
        }
    } else {
        var x3 = Node.Sec(p1, p2);
        // Clamp the secant point when round-off errors
        // shoot it outside the interval
        if (x3 <= p1.X)
             p3 = p1;
         else if (x3 >= p2.X)
             p3 = p2;
         else
             p3 = new Node(x3, F); //Evaluate only if not clamped
        threshold /= 2.0; // Update bisection fallback threshold
    }
    // Check for convergence and return the result
    var eps2 = aTol + rTol * Math.Abs(p3.X);
    if (p3.Y == 0 || p2.X - p1.X <= eps2) return p3.X;
    if (Math.Sign(p1.Y) == Math.Sign(p3.Y)) {
        if (side == 1) {
            // Apply Anderson-Björck correction to the right side
            var m = 1 - p3.Y / p1.Y;
            if (m <= 0) p2.Y /= 2; else p2.Y *= m;
        } else if (!bisection) side = 1;
        p1 = p3;
    } else{
        if (side == -1) {
            // Apply Anderson-Björck correction to the left side
            var m = 1 - p3.Y / p2.Y;
            if (m <= 0) p1.Y /= 2; else p1.Y *= m;
        } else if (!bisection) side = -1;
        p2 = p3;
    }
    // If AB fails to shrink the interval enough
    if (p2.X - p1.X > threshold) {
        bisection = true;          // reset to bisection
```

 doi:10.20944/preprints202603.2190.v1


```
            side = 0;
        }
    }
    return double.NaN; // When failed to converge within maxIterations
```

References

1. Ganchovski N.; Traykov A. Modified Anderson-Björck's method for solving non-linear equations in structural mechanics. *IOP Conference Series: Materials Science and Engineering* **2023**, 1276 (1) 012010, IOP Publishing. doi:10.1088/1757-899X/1276/1/012010.
2. Ganchovski N. Calcpad – a free and open-source software for engineering calculations. Available online: https://github.com/Proektsoftbg/Calcpad (accessed 25 March 2026)
3. Williams J. Roots-Fortran: root solvers for modern Fortran. NASA JSC, Houston, TX, Available online: https://jacobwilliams.github.io/roots-fortran/
4. Pal A.; Holtorf F.; Larsson A.; Loman T.; Utkarsh; Schäfer F.; Qu Q.; Edelman A.; Rackauckas C. NonlinearSolve.jl: High-Performance and Robust Solvers for Systems of Nonlinear Equations in Julia. *ACM TOMS* **2025**, Association for Computing Machinery: NY, USA. https://doi.org/10.1145/3779117
5. Ganchovski N. Structural Analysis by Functional Modelling in the Cloud. *PhD Thesis* **2025**, UACEG, Sofia.
6. Dowell M.; Jarratt P. A modified regula falsi method for computing the root of an equation. *BIT* **1971**, *Vol 11*, pp 168–174. https://doi.org/10.1007/BF01934364
7. Dowell M.; Jarratt P. The "Pegasus" method for computing the root of an equation. *BIT* **1972**, *Vol 12*, pp 503–508. https://doi.org/10.1007/BF01932959
8. Anderson N.; Björck Å. A new high order method of regula falsi type for computing a root of an equation. *BIT* **1973**, *Vol 13*, pp 253–264
9. Oliveira I.; Takahashi R. An Enhancement of the Bisection Method Average Performance Preserving Minmax Optimality. *ACM TOMS* **2021**, *Vol 47*, No 1, Article 5
10. Ridders C. A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, **1979** *Vol 26*, No 11, pp 979-980
11. Brent R. An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal* **1971**, *Vol 14*, Issue 4, pp 422–425
12. Dekker T. Finding a zero by means of successive linear interpolation. *Constructive Aspects of the Fundamental Theorem of Algebra* **1969**, ed B, Dejon and P Henrici (London: Wiley-Interscience) ISBN 978-0-471-20300-1
13. Sabharwal C. Blended Root Finding Algorithm Outperforms Bisection and Regula Falsi Algorithms. *MDPI Mathematics* **2019**, *Vol 7*, Issue 11, Article 1118
14. Kumar R.; Vipan. Comparative Analysis of Convergence of Various Numerical Methods. *Journal of Computer and Mathematical Sciences* **2015**, *Vol 6*, Issue 6, pp 290-297
15. Galdino S. A family of regula falsi root-finding methods. Proceedings of 2011 World Congress on Engineering and Technology, Shanghai, China: IEEE Press, 2011, pp 514-517, ISBN 978-1-61284-365-0
16. Stage S. A. Comments on An Improvement to the Brent's Method. *International Journal of Experimental Algorithms* (IJEA) **2013**, *Vol 4*, Issue 1, pp 1-16
17. Swift A.; Lindfield G. R. Comparison of a continuation method with Brent's method for the numerical solution of a single nonlinear equation. *The Computer Journal* **1978**, *Vol 21*, Issue 4, pp 359–362
18. Suhadolnik A. Combined bracketing methods for solving nonlinear equations. *Applied Mathematics Letters* **2012**, *Vol 25*, Issue 11, pp 1755-1760, ISSN 0893-9659
19. Sikorski K. Bisection is optimal. *Numerische Mathematik* **1982**, *Vol 40*, pp 111–117 https://doi.org/10.1007/BF01459080
20. Press W.; Teukolsky S.; Vetterling W.; Flannery B. *Numerical Recipes: The Art of Scientific Computing*, 3rd Edition. Cambridge University Press: USA, 2007
21. Verzani, J. *Roots.jl: Root Finding Functions for Julia*. 2020. Available online: https://github.com/JuliaMath/Roots.jl (accessed 25 March 2026).
22. Brun, R.; Rademakers, F. ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sept. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 1997, pp 81-86.

 doi:10.20944/preprints202603.2190.v1