algorithms

MDPI

Article

Improvements to 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, 1164 Sofia, Bulgaria
[2] JuliaHub, Cambridge, MA 02138, USA; oscar.smith@juliahub.com (O.S.); chris.rackauckas@juliahub.com (C.R.)
[3] CSAIL, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[4] Department of Computer Science, New Bulgarian University, 1618 Sofia, Bulgaria; lptomov@nbu.bg
[5] Department of Structural Mechanics, University of Architecture, Civil Engineering and Geodesy, 1164 Sofia, Bulgaria; alex_fce@uacg.bg
* Correspondence: proektsoft.bg@gmail.com

Abstract

The Modified Anderson–Björck method is a new, robust, and efficient bracketing root-finding algorithm. It combines bisection with the Anderson–Björk 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, such as Ridders, Brent and ITP. In this paper, we propose further improvements to this method and perform some additional analysis and benchmarks of its behavior and performance.

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

Check for updates

Academic Editor: Ming-Feng Ge

Received: 25 March 2026
Revised: 14 April 2026
Accepted: 21 April 2026
Published: 24 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Iterative algorithms are generally fast but can easily diverge if an improper starting point is selected. Bracketing algorithms are more reliable in this respect. If the function is continuous in the interval [a, b] and has opposite signs at both ends, according to Bolzano's theorem, at least one root exists inside [a, b], and the bracketing method is guaranteed to find it within a certain precision. Bisection is the simplest method, and it is 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 using 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 [1], and Ridder's [2] can provide significantly better performance, but for poorly behaved functions, they can perform significantly worse than bisection. Therefore, stability and performance are contradictory goals and achieving both is a challenging task. Some authors have noted that this can be done by combining bisection with a faster algorithm like false-position [3], secant or inverse-quadratic interpolation. Examples of these combinations include Brent's method [4] and the ITP [5] method. Also, initial bisection steps can boost the performance of false-position and similar methods by bringing the starting point

to more favorable conditions [6,7]. Although it is obtained through a combination of linear algorithms, the resulting algorithm shows nonlinear (bilinear) behavior and better convergence than using both algorithms 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.

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

On that basis, a new hybrid algorithm was proposed in 2023 [6] 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 following equation:

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

where y_3 is the function value at the 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 original algorithm has been included in the Calcpad [8] and RootFortan [9] libraries. Although the algorithm has shown superior performance compared to others in tests, and later in practice, we have identified some drawbacks and opportunities for further improvements.

3. Drawbacks of the Original Algorithm

The original algorithm has several shortcomings that can be addressed:

1. The deviation factor k in Equation (2) has a fixed value of $k = 0.25$ as a good average estimate, experimentally tuned on a larger set of functions. However, this is not optimal because it does not account for the properties of the function in the selected interval.
2. Although the possibility of a redundant switch to AB was discussed in [6], not much research 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 certain cases, it may result in unnecessary AB steps before the algorithm recognizes that it is not converging as intended.
3. 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 near the root, but with large ordinates at endpoints ($>1/\varepsilon$). This can cause the interpolation point to crawl very slowly along the abscissa, so the step may become lower than ε, even if we are still too far from the root.

4. The benchmarks in the original paper [6] were based on the number of iterations with a factor of two for Ridder's method. However, the number of function evaluations is more closely related to the actual performance, especially when the function is computationally expensive.
5. In the original paper, we used 21 functions to benchmark the method performance. This set is very limited, which can lead to some distortion in the results. Therefore, the set may be considered insufficient to derive reliable conclusions. It may turn out 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 [7] with a set of 50 functions.

4. Improvements to the modAB Algorithm

In this paper, we propose a new version of the modAB algorithm, aiming for it 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.

4.1. Adaptive Value of Deviation Factor k

For highly curved functions like the one in Figure 1, symmetry plays a key role in the behavior of linear interpolation algorithms. If the endpoint ordinates are highly asymmetric, as shown in the left plot, it is possible 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 also becomes smaller. If the endpoint values are nearly symmetric, the behavior improves significantly, even when both functions have the same deviation from the straight line.

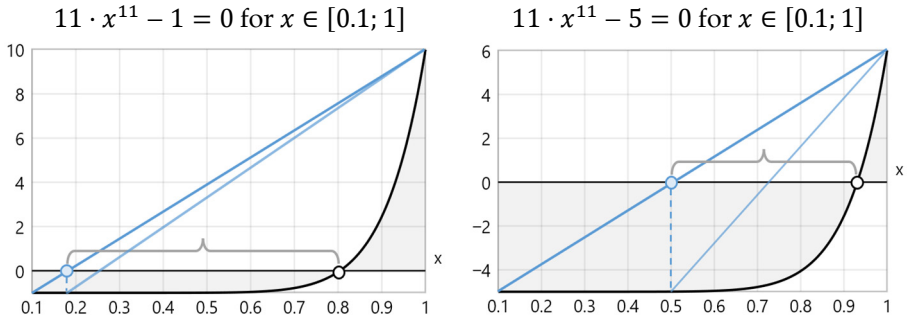


Figure 1. Plot of function $f(x) = 11 \cdot x^{11} - c$, for $c = 1$ and 5.

For that reason, it is reasonable to include symmetry in the method's switching criterion, and the factor k seems to be the right place to do so. For unsymmetric endpoints, we can keep k lower, forcing the algorithm to take more bisection steps. For symmetric endpoints, we can make k larger and allow the algorithm to switch faster to AB, since we expect it to be more favorable here. However, this is highly dependent on the shape of the function. Therefore, tuning it too much for specific types of functions is not justified. Instead, we can 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 following 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 do we take this into account

in the deviation factor k? If we look closely at the above functions, we can see that the relationship between symmetry and convergence is not linear. Higher symmetry leads to both a shorter path and longer step size, which is a quadratic relationship. 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 an allowance to grow up to 1 for fully symmetric endpoints.

4.2. Adaptive Bisection Fallback Criterion

Instead of using a fixed number of expected iterations, we can apply a more precise criterion by measuring the actual performance of the AB algorithm after switching. A possible 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 length

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

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

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

where C is a safety factor. Let $W(t) = b_t - a_t$ denote 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 switching without a safety factor causes the algorithm to oscillate rapidly between methods, significantly degrading the performance. Therefore, 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 between three and four. Larger values caused the algorithm to waste too many iterations on AB. Therefore, 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 switching to AB as $T_0 = CW_0$. Then, we will update it on each AB iteration t using the equation $T_t = T_{t-1}/2$.

To demonstrate how the new switching criteria works, 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 in Figure 2.

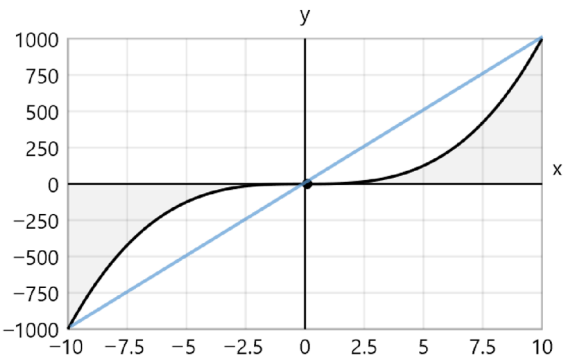


Figure 2. Plot of function $f(x) = x^3 - 0.001$, for $x \in [-10, 10]$.

Both the original and proposed algorithms switch to AB at the first iteration, finding themselves in a position to deal with a highly nonlinear and unsymmetric problem. After

that, they show very different behavior, as summarized in Table 1 below. This demonstrates that using fixed switching criteria is not optimal and can possibly slow down the algorithm.

Table 1. Comparison of switching criteria in the original and the proposed methods.

Algorithm	Iteration Number			
	Switch to AB	**Fallback to Bisection**	**Switch to AB**	**Converge and Quit**
The original modAB 2023 [6] 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 a more robust and reliable bisection-style check $b_n - a_n < \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \cdot |x_n|$, 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 the AB corrections always cause the interval to shrink from both sides. However, for some functions, shrinking from the other side may require one or two extra steps, after the solution has converged from one side. Nevertheless, this trade-off is justified by the improved robustness and reliability of the method.

Additionally, we can separate the convergence checks along x and y. Since the x-check does not depend on the y-value, we can move it to the top and thus save one function evaluation.

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):

Algorithm 1: The improved modAB algorithm.

1. Calculate points p_1: x_1, $y_1 = f(x_1)$ and p_2: x_2, $y_2 = f(x_2)$
2. initialize "method" to "bisection" and "side" to "none"
3. for iter = 1 to *maxIterations*
4. if "method" is "bisection" then:
5. calculate x_3 as the midpoint: $x_3 = (x_1 + x_2)/2$
6. else if method is "Anderson-Björck" then:
7. calculate x_3 as the secant point: $x_3 = (x_1{*}y_2 - y_1{*}x_2)/(y_2 - y_1)$
8. end
9. if the x-convergence check $x_2 - x_1 \leq \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}}{*}|x_3|$ is satisfied then:
10. stop and return x_3 as a result (*avoids redundant evaluation of* $y_3 = f(x_3)$)
11. end
12. if "method" is "bisection" then:
13. calculate p_3: x_3, $y_3 = f(x_3)$
14. calculate the average ordinate by the equation: $y_m = (y_1 + y_2)/2$
15. calculate r to Equation (4) and $k = r^2$
16. if the function is close enough to linear, satisfying that _ $|y_m - y_3| < k\,(|y_3| + |y_m|)$ then:
17. change the method to "Anderson-Björck"
18. end
19. else if method is "Anderson-Björck" then:

Algorithm 1: Cont.

```
        calculate p3: x3, y3 = f(x3) and clamp it between p1 and p2 _
        (in case floating point round-off errors shoot it outside the interval)
    end
    if the y-convergence check y3 = 0 is satisfied then:
        stop and return x3 as a result
    end
    if y1 and y3 have equal signs then:
        if "method" is "Anderson-Björck" then: (apply AB correction)
            if "side" is "right" then
                calculate m = 1 − y3/y1
                if m ≤ 0 then: divide y2 by 2 else: multiply y2 by m
            else:
                change "side" to "right"
            end
        end
        for both methods, shrink the interval from left by setting p1 = p3
    else
        if "method" is "Anderson-Björck" then:
            if "side" is "left" then:
                calculate m = 1 − y3/y2
                if m ≤ 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 p2 = p3
    end
    if AB fails to shrink the interval more than bisection would, then:
        reset the method back to "bisection"
    end
end
return error, if failed to converge for the expected number of iterations
```

4.5. Convergence

Since the modAB algorithm represents a modification of Anderson–Björck's method [10], guarded by bisection, it should naturally inherit its superlinear convergence, estimated to be between 1.7 and 1.8. This assumption corresponds well to the results from the numerical experiments presented below. The proposed changes are aimed mostly at improving the worst-case scenarios where the original AB algorithm struggles to converge. However, more rigorous convergence analysis will be performed in future and published in a separate paper.

4.6. Practical Use and Implementation in Software

The source code of the algorithm is listed in Appendix A. It has also been translated into Julia, Python, Cython, C, C++ and Zig languages. The modAB algorithm is included in Calcpad v7.6.2 [8], SciML/NonlinearSolve.jl v4.17.1 [11], JuliaMath/Roots.jl v2.2.11 [12], and ROOT.CERN v6-40-00-rc1 [13] libraries for scientific computations. It can be used in Python v3.x via the PyModAB v1.0.3 [14] package. For all implementations, modAB

demonstrated superior performance compared to the existing algorithms, according to the benchmarks.

5. Numerical Experiments

The improved modAB algorithm is benchmarked against the original version and seven classical algorithms: bisection, false-position, Illinois [1], Anderson–Björck [9], ITP [5], Ridders [2] and Brent [4,15]. For this purpose, they are also implemented in C# programming language.

An extended set of 92 functions is used for the benchmarks. To avoid bias, they are collected from papers by authors of other algorithms and independent comparative studies as follows:

- Sérgio Galdino [16]: $f_{01} - f_{30}$
- Custom functions [7]: $f_{31} - f_{32}$
- Steven A. Stage [17]: $f_{33} - f_{40}$
- Swift & Lindfield [18]: $f_{41} - f_{50}$
- Alojz Suhadolnik [19]: $f_{51} - f_{62}$
- Oliveira & Takahashi [5]: $f_{63} - f_{83}$
- SciML Benchmarks suite [11]: $f_{84} - f_{92}$

Plots of all functions with equations and root values are presented in the Supplementary File: Test Functions.pdf. The benchmarks are based on the number of function evaluations, which are more indicative of the actual performance than the number of iterations, especially for computationally expensive functions. The results are presented in Table 2.

Table 2. Benchmark results for solving $f_{nn}(x) = 0$ in the interval $[a; b]$.

Function	Interval		Method								
	a	***b***	**bs**	**fp**	**ill**	**AB**	**ITP**	**Rid**	**Bre**	**modAB [1]**	**modAB [2]**
$f_{01}(x) = x^3 - 1$	0.5	1.5	**3**	34	10	9	10	4	10	**3**	**3**
$f_{02}(x) = x^2\,(x^2/3 + \mathrm{sqrt}(2) \cdot \sin(x)) - \mathrm{sqrt}(x)/18$	0.1	1	48	87	**12**	**12**	39	16	13	**12**	**12**
$f_{03}(x) = 11x^{11} - 1$	0.1	1	48	108	16	23	50	14	**12**	13	15
$f_{04}(x) = x^3 + 1$	−1.8	0	49	50	13	11	12	12	**10**	**10**	**10**
$f_{05}(x) = x^3 - 2x - 5$	2	3	47	32	9	**8**	51	14	**8**	10	12
$f_{06}(x) = 2x \cdot \exp(-5) + 1 - 2 \cdot \exp(-5x)$	0	1	49	30	**10**	**10**	13	14	**10**	12	**10**
$f_{07}(x) = 2x \cdot \exp(-10) + 1 - 2 \cdot \exp(-10x)$	0	1	49	31	13	**11**	15	14	12	13	**11**
$f_{08}(x) = 2x \cdot \exp(-20) + 1 - 2 \cdot \exp(-20x)$	0	1	49	31	13	13	17	**12**	**12**	13	**12**
$f_{09}(x) = (1 + (1-5)^2) \cdot x^2 - (1-5x)^2$	0	1	49	17	12	**9**	48	16	10	12	11
$f_{10}(x) = (1 + (1-10)^2) \cdot x^2 - (1-10x)^2$	0	1	49	14	10	**9**	12	16	10	**9**	**9**
$f_{11}(x) = (1 + (1-20)^2) \cdot x^2 - (1-20x)^2$	0	1	49	12	10	**8**	10	16	9	9	9
$f_{12}(x) = x^2 - (1-x)^5$	0	1	49	41	10	**9**	11	14	**9**	12	11
$f_{13}(x) = x^2 - (1-x)^{10}$	0	1	49	83	12	**10**	49	16	**10**	12	12
$f_{14}(x) = x^2 - (1-x)^{20}$	0	1	49	171	**12**	**12**	50	16	13	13	**12**
$f_{15}(x) = (1 + (1-5)^4) \cdot x - (1-5x)^4$	0	1	49	9	9	**7**	10	12	8	9	9
$f_{16}(x) = (1 + (1-10)^4) \cdot x - (1-10x)^4$	0	1	49	**6**	7	**6**	9	12	7	8	8
$f_{17}(x) = (1 + (1-20)^4) \cdot x - (1-20x)^4$	0	1	49	**6**	7	**6**	9	12	7	8	8
$f_{18}(x) = \exp(-5x) \cdot (x-1) + x^5$	0	1	48	81	10	**8**	37	14	9	11	11
$f_{19}(x) = \exp(-10x) \cdot (x-1) + x^{10}$	0	1	48	202	14	**9**	11	16	**9**	13	12
$f_{20}(x) = \exp(-20x) \cdot (x-1) + x^{20}$	0	1	48	202	22	**10**	12	16	13	13	14

Table 2. *Cont.*

Function	Interval		Method								
	a	***b***	**bs**	**fp**	**ill**	**AB**	**ITP**	**Rid**	**Bre**	**modAB [1]**	**modAB [2]**
$f_{21}(x) = x^2 + \sin(x/5) - 1/4$	0	1	49	34	10	**9**	10	12	11	12	**9**
$f_{22}(x) = x^2 + \sin(x/10) - 1/4$	0	1	48	33	11	**9**	10	12	10	**9**	**9**
$f_{23}(x) = x^2 + \sin(x/20) - 1/4$	0	1	48	32	12	**9**	10	12	10	**9**	**9**
$f_{24}(x) = (x+2) \cdot (x+1) \cdot (x-3)^3$	2.6	4.6	**48**	202	91	108	51	80	137	49	**48**
$f_{25}(x) = (x-4)^5 \cdot \log(x)$	3.6	5.6	**48**	202	174	188	51	62	118	49	**48**
$f_{26}(x) = (\sin(x) - x/4)^3$	2	4	**48**	202	91	108	51	78	137	49	**48**
$f_{27}(x) = (81 - p(x) \cdot (108 - p(x) \cdot (54 - p(x) \cdot (12 - p(x))))) \cdot \text{sign}\,(p(x) - 3)$, where $p(x) = x + 1.11111$	1	3	**14**	202	35	38	22	24	31	**14**	**14**
$f_{28}(x) = \sin(x - 7.143)^3)$	7	8	**46**	202	87	102	50	70	131	47	**46**
$f_{29}(x) = \exp((x-3)^5) - 1$	2.6	4.6	12	202	59	46	**3**	18	30	12	12
$f_{30}(x) = \exp((x-3)^5) - \exp(x-1)$	4	5	47	202	52	57	50	16	14	14	**13**
$f_{31}(x) = \pi - 1/x$	0.05	5	51	192	15	**7**	53	16	12	13	12
$f_{32}(x) = 4 - \tan(x)$	0	1.5	48	92	14	**11**	51	16	13	13	14
$f_{33}(x) = \cos(x) - x^3$	0	4	50	200	15	15	**11**	18	15	13	12
$f_{34}(x) = \cos(x) - x$	−11	9	53	11	10	**9**	11	16	10	10	11
$f_{35}(x) = \text{sqrt}(\text{abs}(x - 2/3)) \cdot \text{if}(x \leq 2/3; 1; -1) - 0.1$	−11	9	53	15	16	13	54	24	**11**	18	17
$f_{36}(x) = \text{abs}(x - 2/3)^{0.2} \cdot \text{if}(x \leq 2/3; 1; -1)$	−11	9	53	**44**	47	54	46	54	**44**	54	55
$f_{37}(x) = (x - 7/9)^3 + (x - 7/9) \cdot 10^{-3}$	−11	9	52	202	25	27	**13**	24	31	18	16
$f_{38}(x) = \text{if}(x \leq 1/3; -0.5; 0.5)$	−11	9	**53**	**53**	61	61	**53**	56	**53**	54	**53**
$f_{39}(x) = \text{if}(x \leq 1/3; -10^{-3}; 1 - 10^{-3})$	−11	9	**53**	202	172	172	55	84	83	54	**53**
$f_{40}(x) = \text{if}(x = 0; 0.0; 1/(x - 2/3))$	−11	9	53	**8**	100	100	55	60	55	54	53
$f_{41}(x) = 2x \cdot \exp(-5) - 2 \cdot \exp(-5x) + 1$	0	10	52	33	14	**13**	17	14	**13**	15	**13**
$f_{42}(x) = (x^2 - x - 6) \cdot (x^2 - 3x + 2)$	0	π	50	23	12	**9**	16	18	10	13	12
$f_{43}(x) = x^3$	−1	1.5	**50**	202	94	113	52	84	124	51	**50**
$f_{44}(x) = x^5$	−1	1.5	**50**	202	183	202	52	56	127	51	**50**
$f_{45}(x) = x^7$	−1	1.5	**50**	202	202	202	52	52	135	51	**50**
$f_{46}(x) = (\exp(-5x) - x - 0.5)/x^5$	0.09	0.7	48	202	25	23	50	**14**	15	**14**	**14**
$f_{47}(x) = 1/\text{sqrt}(x) - 2 \cdot \log(5 \cdot 10^3 \cdot \text{sqrt}(x)) + 0.8$	0.0005	0.5	48	155	16	**13**	49	18	16	16	16
$f_{48}(x) = 1/\text{sqrt}(x) - 2 \cdot \log(5 \cdot 10^7 \cdot \text{sqrt}(x)) + 0.8$	0.0005	0.5	48	44	15	**13**	20	18	16	19	14
$f_{49}(x) = \text{if}\,(x \leq 0; -x^3 - x - 1; x^{1/3} - x - 1)$	−1	1	49	26	12	**9**	12	12	11	12	12
$f_{50}(x) = x^3 - 2x - x + 3$	−3	2	50	41	15	13	16	14	12	12	**11**
$f_{51}(x) = \log(x)$	0.5	5	50	28	12	**9**	12	12	10	11	10
$f_{52}(x) = (10 - x) \cdot \exp(-10x) - x^{10} + 1$	0.5	8	51	202	38	20	54	18	17	**15**	**15**
$f_{53}(x) = \exp(\sin(x)) - x - 1$	1.0	4	49	32	**11**	**11**	12	18	14	13	13
$f_{54}(x) = 2 \cdot \sin(x) - 1$	0.1	$\pi/3$	48	16	9	**8**	12	12	**8**	10	11
$f_{55}(x) = (x-1) \cdot \exp(-x)$	0.0	1.5	49	70	13	10	12	**4**	11	10	10
$f_{56}(x) = (x-1)^3 - 1$	1.5	3	48	57	13	**10**	11	14	11	11	11
$f_{57}(x) = \exp(x^2 + 7x - 30) - 1$	2.6	3.5	47	202	21	31	50	14	12	12	**11**
$f_{58}(x) = \text{atan}(x) - 1$	1.0	8	50	26	12	**9**	13	12	10	12	10
$f_{59}(x) = \exp(x) - 2x - 1$	0.2	3	49	147	16	14	**12**	14	13	**12**	**12**
$f_{60}(x) = \exp(-x) - x - \sin(x)$	0.0	2	50	15	9	**8**	11	16	8	9	9
$f_{61}(x) = x^2 - \sin(x)^2 - 1$	−1	2	49	33	13	13	13	18	**11**	12	13
$f_{62}(x) = \sin(x) - x/2$	$\pi/2$	π	48	32	10	**8**	10	14	9	10	10
$f_{63}(x) = x \cdot \exp(x) - 1$	−1	1	49	30	12	**10**	12	**10**	**10**	11	11
$f_{64}(x) = \tan(x - 1/10)$	−1	1	50	40	**8**	**8**	11	12	9	9	**8**

Table 2. *Cont.*

Function	Interval		Method								
	a	***b***	**bs**	**fp**	**ill**	**AB**	**ITP**	**Rid**	**Bre**	**modAB [1]**	**modAB [2]**
$f_{65}(x) = \sin(x) + 0.5$	−1	1	49	11	9	**8**	11	16	**8**	9	9
$f_{66}(x) = 4x^5 + x^2 + 1$	−1	1	49	33	13	11	15	16	**10**	13	13
$f_{67}(x) = x + x^{10} - 1$	−1	1	49	50	13	**11**	51	12	12	14	12
$f_{68}(x) = \pi^x - e$	−1	1	49	15	10	**8**	11	10	**8**	9	9
$f_{69}(x) = \log(\text{abs}(x - 10/9))$	−1	1	50	64	13	**9**	10	12	10	10	10
$f_{70}(x) = 1/3 + \text{sign}(x) \cdot \text{abs}(x)^{1/3} + x^3$	−1	1	50	13	13	**11**	16	16	**11**	12	17
$f_{71}(x) = (x + 2/3)/(x + 101/100)$	−1	1	49	202	16	**7**	51	16	13	10	8
$f_{72}(x) = (x \cdot 10^6 - 1)^3$	−1	1	50	**3**	**3**	**3**	51	6	139	51	50
$f_{73}(x) = \exp(x) \cdot (x \cdot 10^6 - 1)^3$	−1	1	50	202	95	114	51	**6**	141	51	50
$f_{74}(x) = (x - 1/3)^2 \cdot \text{atan}(x - 1/3)$	−1	1	**50**	202	93	112	51	82	142	51	**50**
$f_{75}(x) = \text{sign}(3x - 1) \cdot (1 - \text{sqrt}(1 - (3x - 1)^2/81))$	−1	1	27	202	28	37	**23**	38	66	27	27
$f_{76}(x) = \text{if}(x > (1 - 10^6)/10^6; (1 + 10^6)/10^6; -1)$	−1	1	49	49	**37**	**37**	50	52	49	50	39
$f_{77}(x) = \text{if}\ (x \neq 1/21; 1/(21\,x - 1); 0)$	−1	1	50	**14**	95	95	51	66	53	51	50
$f_{78}(x) = x^2/4 + \text{ceiling}(x/2) - 0.5$	−1	1	50	50	56	55	50	40	51	51	**9**
$f_{79}(x) = \text{ceiling}(10x - 1) + 0.5$	−1	1	50	48	52	52	47	38	47	15	**14**
$f_{80}(x) = x + \sin(x \cdot 10^6)/10 + 10^{-3}$	−1	1	50	24	22	22	25	26	**21**	25	25
$f_{81}(x) = \text{if}\ (x > -1; 1 + \sin(1/(x + 1)); -1)$	−1	1	49	52	15	**14**	51	40	51	20	**14**
$f_{82}(x) = 202x - 2 \cdot \text{floor}((2x + 10^{-2})/(2 \cdot 10^{-2})) - 0.1$	−1	1	50	**3**	**3**	**3**	13	6	4	5	5
$f_{83}(x) = (202x - 2 \cdot \text{floor}((2x + 10^{-2})/(2 \cdot 10^{-2})) - 0.1)^3$	−1	1	**50**	202	73	70	51	70	125	51	**50**
$f_{84}(x) = (x - 1) \cdot (x - 2) \cdot (x - 3) \cdot (x - 4) \cdot (x - 5) - 0.05$	0.5	5.5	49	66	9	**7**	53	20	8	8	8
$f_{85}(x) = \sin(x) - 0.5x - 0.3$	−10.0	10.0	52	14	13	**11**	47	20	12	12	**11**
$f_{86}(x) = \exp(x) - 1 - x - x^2/2 - 0.005$	−2.0	2.0	51	202	19	19	22	20	16	**15**	**15**
$f_{87}(x) = 1/(x - 0.5) - 2 - 0.05$	0.6	2.0	48	136	14	**5**	51	16	12	8	9
$f_{88}(x) = \log(x) - x + 2 - 0.05$	0.1	3.0	50	29	13	**11**	17	16	13	14	14
$f_{89}(x) = \sin(20x) + 0.1\,x - 0.1$	−4.0	5.0	51	17	17	15	17	20	17	16	**13**
$f_{90}(x) = x^3 - 2x^2 + x - 0.025$	−1.0	2.0	51	175	**13**	**13**	52	18	15	**13**	**13**
$f_{91}(x) = x \cdot \sin(1/x) - 0.1 - 0.01$	0.01	1.0	49	13	13	**11**	39	14	13	13	13
$f_{92}(x) = x^3 - 0.001$	−10	10	53	202	**24**	202	55	28	**24**	42	25
Total number of evaluations			4410	8132	2907	3095	2798	2256	2880	1857	1741
Average number of evaluations			47.9	88.4	31.6	33.6	30.4	24.5	31.3	20.2	18.9
Relative to modAB [2]			2.53	4.67	1.67	1.78	1.61	1.30	1.65	1.07	1.00
Maximal number of evaluations			53	202	202	202	55	84	142	54	55
Geometric mean			46.3	52.0	19.2	17.5	23.9	19.1	18.3	15.9	15.0
Best at (number of functions)			13	8	10	52	6	5	22	12	38
Worst at (number of functions)			38	43	6	7	3	1	1	0	1
Median			49.0	48.5	13.0	11.0	22.0	16.0	12.0	13.0	12.0
Standard deviation			7.6	79.0	41.0	48.2	19.0	20.5	40.2	16.0	15.1
Standard error			0.80	8.23	4.27	5.03	1.98	2.13	4.19	1.67	1.58
			bs	fp	ill	AB	ITP	Rid	Bre	modAB [1]	modAB [2]

Legend: bs—bisection; Bre—Brent; fp—false-position; Rid—Ridders; ill—Illinois; AB—Anderson–Björck; ITP—Interpolate, Truncate, Project; modAB [1]—Modified Anderson–Björck—the original 2022 version; modAB [2]—Modified Anderson–Björck (the proposed version). Note: The best algorithm for each function is **highlighted**.

Additionally, we have benchmarked the time performance of the above algorithms for the same set of functions using the BenchmarkDotNet tool [20]. The results are presented in Table 3.

Table 3. Total execution times * for all functions in us (microseconds).

Algorithm	Mean	Error	StdDev	Relative
Bisection	86.57 us	0.832 us	0.738 us	1.75
FalsePosition	321.77 us	3.561 us	2.974 us	6.51
Illinois	104.75 us	1.353 us	1.265 us	2.12
Anderson–Björck	122.19 us	1.205 us	1.127 us	2.47
ITP	193.37 us	1.604 us	1.339 us	3.92
Ridders	66.61 us	1.275 us	1.309 us	1.35
Brent	119.56 us	1.213 us	1.134 us	2.42
modAB [1]	51.58 us	1.013 us	1.352 us	1.04
modAB [2]	49.39 us	0.323 us	0.269 us	1.00

* BenchmarkDotNet v0.15.8, .NET 10.0.5 x64 RyuJIT x86-64-v4; Windows 11 (10.0.26200.8037/25H2/2025Update/HudsonValley2); Intel Core i7-1065G7 CPU 1.30 GHz eight logical and four physical cores + 16 GB RAM; [1] the original version; [2] the proposed version.

If we plot both the number of evaluations and benchmark times side by side, we can see that the time differences are greater for all algorithms, except for bisection, as displayed in Figure 3.

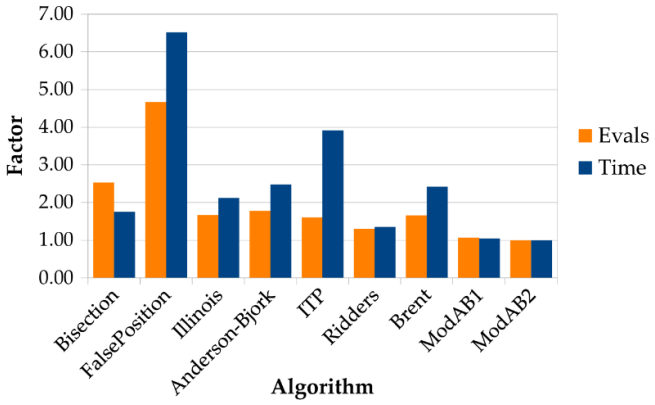


Figure 3. Combined benchmark results.

For computationally expensive functions, the number of evaluations governs because most of the time is spent on the function itself. For lightweight functions, the computational overhead of the algorithm can be important. Bisection is the simplest method, so it shows better results in time than in number of evaluations. ITP is on the opposite side. It is very efficient in terms of evaluation count, but computationally intensive. Its larger overhead per iteration downgrades its performance in the time domain. ModAB uses simple linear interpolation (with endpoint corrections) and bisection. This makes it very lightweight, and introduces very little overhead per iteration, especially when bisection is dominant. This is an advantage when multiple simple equations must be solved in a tight loop.

6. Discussion

The results show an improvement of about 7% in the proposed modAB algorithm compared to the original one, requiring 1741 total evaluations vs. 1857, respectively. If we keep the same convergence check, the effect of improving the method switching and fallback criteria is about 10% (1672 total evaluations). Therefore, the new convergence check slows down the algorithm by 3%; however, providing robustness is always more important than speed. Compared to the original Anderson–Björck algorithm, there is an improvement of about 1.78 times. The new modAB algorithm also outperforms Ridder's by 1.3 times, ITP by 1.61 times and Brent by 1.65 times.

Compared to the original 2023 version, the proposed changes save 116 evaluations. Also, 45 functions are improved, and 10 functions are made worse, mostly by between one and two evaluations, except for f70, for which five evaluations are added. The greatest improvement is observed for f78, where the updated linearity check allows AB to take over in time, and save 42 evaluations. The new adaptive bisection fallback provides the most help to f92, saving 17 evaluations.

For well-behaved functions, the new algorithm often shows the best performance (38 functions) or remains between one and two evaluations behind the winner. For poorly behaved functions, the guarding role of bisection appears to work, providing no more than 55 evaluations in the worst case. This is probably due to spurious switching from bisection to AB and the fallback. It can also be noted that after extending the number of test functions, the relationships between the efficiencies of the separate algorithms are preserved. This indicates that the conclusions previously derived in [6] were not distorted by the chosen set of functions.

By analyzing the standard deviation and error in Tables 2 and 3, we can observe that the modAB algorithm provides more uniform and consistent behavior for various types of functions.

7. Conclusions

The proposed method is suitable for use as a general-purpose algorithm for solving nonlinear equations in numerical libraries and software where guaranteed convergence, robustness and high performance are required but no preliminary information about the specific properties of the function and its derivatives is available.

Supplementary Materials: The following supporting information can be downloaded at https://www.mdpi.com/article/10.3390/a19050332/s1: [CSharp code] folder: complete C# source code; Test Functions.pdf file: equations and plots of functions with the corresponding roots; Test Results.xlsx file: detailed benchmark results, including root values and function values.

Author Contributions: Project administration, N.G.; methodology, N.G.; software, N.G., O.S. and C.R.; validation, O.S., C.R. and L.T.; writing—original draft preparation, N.G.; writing—review and editing, O.S., C.R., L.T. and A.T. 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 Materials. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors express their gratitude to the reviewers for providing their valuable comments and recommendations, which significantly contributed to the improvement of the final version of this article.

Conflicts of Interest: Author Nedelcho Ganchovski was employed by the company Proektsoft EOOD. Authors Oscar Smith and Christopher Rackauckas were employed by the company JuliaHub. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AB	Anderson–Björck
modAB	Modified Anderson–Björck
ITP	Interpolate, Truncate, Project

Appendix A

Table A1. 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);
}
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 the improved modAB
// algorithm. 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
var side = 0;                      // Store the side that moved last:
// -1 for left, 1 for right, 0 for none
var threshold = p2.X - p1.X; // Threshold to reset to bisection
const double C = 16; // Safety factor of 4 iterations behind the threshold
for (int i = 1; i <= MaxIterations; ++i) {
var x3 = bisection ? Node.Mid(p1, p2) : Node.Sec(p1, p2);
// Check for x-convergence and return the result
if (p2.X - p1.X <= aTol + rTol * Math.Abs(x3)) return x3;
Node p3;
if (bisection) {
p3 = new Node(x3, F);
double y1 = p1.Y, y2 = p2.Y;
var ym = 0.5 * (y1 + y2);
var r = 1 - Math.Abs(ym / (y2 - y1)); // Symmetry factor
var k = r * r;                        // Deviation factor
// If function is close enough to linear
if (Math.Abs(ym - p3.Y) < k * (Math.Abs(p3.Y) + Math.Abs(ym))) {
bisection = false; // switch to false-position
threshold = (p2.X - p1.X) * C;
}
} else {
// Clamp the secant point to the interval [p1.X, p2.X]
// Otherwise, for very flat functions, floating-point
// round-off errors can shoot the point outside
if (x3 <= p1.X)
p3 = p1;
else if (x3 >= p2.X)
p3 = p2;
```

Table A1. *Cont.*

```
55.     else
56.     p3 = new Node(x3, F); // Evaluate only if not clamped
57.
58.     threshold * = 0.5;
59.     }
60.     // Check for y-convergence and return the result
61.     if (p3.Y == 0) return x3;
62.     if (Math.Sign(p1.Y) == Math.Sign(p3.Y)) {
63.     if (side == 1) {
64.     var m = 1 - p3.Y / p1.Y; // Apply Anderson-Björck correction
65.     p2.Y *= m <= 0 ? 0.5 : m; // to the right side
66.     }
67.     else if (!bisection) side = 1;
68.     p1 = p3;
69.     } else {
70.     if (side == -1) {
71.     var m = 1 - p3.Y / p2.Y; // Apply Anderson-Björck correction
72.     p1.Y *= m <= 0 ? 0.5 : m; // to the left side
73.     }
74.     else if (!bisection) side = -1;
75.     p2 = p3;
76.     }
77.     // If Anderson-Björck fails to shrink the interval enough
78.     if (p2.X - p1.X > threshold) {
79.     bisection = true; // reset to bisection
80.     side = 0;
81.     }
82.     }
83.     return double.NaN; // When failed to converge within maxIterations
84.     }
```

References

1. Dowell, M.; Jarratt, P. A modified regula falsi method for computing the root of an equation. *BIT* **1971**, *11*, 168–174. [CrossRef]
2. Ridders, C. A new algorithm for computing a single root of a real continuous function. *IEEE Trans. Circuits Syst.* **1979**, *26*, 979–980. [CrossRef]
3. Sabharwal, C. Blended Root Finding Algorithm Outperforms Bisection and Regula Falsi Algorithms. *Mathematics* **2019**, *7*, 1118. [CrossRef]
4. Brent, R. An algorithm with guaranteed convergence for finding a zero of a function. *Comput. J.* **1971**, *14*, 422–425. [CrossRef]
5. Oliveira, I.; Takahashi, R. An Enhancement of the Bisection Method Average Performance Preserving Minmax Optimality. *ACM Trans. Math. Softw.* **2021**, *47*, 5. [CrossRef]
6. Ganchovski, N.; Traykov, A. Modified Anderson-Björck's method for solving non-linear equations in structural mechanics. *IOP Conf. Ser. Mater. Sci. Eng.* **2023**, *1276*, 012010. [CrossRef]
7. Ganchovski, N. Structural Analysis by Functional Modelling in the Cloud. Ph.D. Thesis, UACEG, Sofia, Bulgaria, 2025.
8. Ganchovski, N. Calcpad—A Free and Open-Source Software for Engineering Calculations. Available online: https://calcpad.eu/ (accessed on 16 April 2026).
9. Williams, J. *Roots-Fortran: Root Solvers for Modern Fortran*; NASA JSC: Houston, TX, USA, 2025; Available online: https://jacobwilliams.github.io/roots-fortran/ (accessed on 25 March 2026).
10. Anderson, N.; Björck, Å. A new high order method of regula falsi type for computing a root of an equation. *BIT* **1973**, *13*, 253–264. [CrossRef]
11. 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 Trans. Math. Softw.* **2026**, *52*, 1. [CrossRef]
12. Verzani, J. Roots.jl: Root Finding Functions for Julia. 2020. Available online: https://github.com/JuliaMath/Roots.jl (accessed on 25 March 2026).
13. Brun, R.; Rademakers, F. ROOT—An Object Oriented Data Analysis Framework. *Nucl. Inst. Meth. Phys. Res. A* **1997**, *389*, 81–86. [CrossRef]

14. PyModAB—Fast and Robust Root-Finding Library Using the Modified Anderson-Bjork Method. Available online: https://pypi.org/project/pymodab/ (accessed on 10 April 2026).
15. Press, W.; Teukolsky, S.; Vetterling, W.; Flannery, B. *Numerical Recipes: The Art of Scientific Computing*, 3rd ed.; Cambridge University Press: Cambridge, MA, USA, 2007.
16. Galdino, S. A family of regula falsi root-finding methods. In *Proceedings of 2011 World Congress on Engineering and Technology*; IEEE Press: Shanghai, China, 2011; pp. 514–517.
17. Stage, S.A. Comments on An Improvement to the Brent's Method. *Int. J. Exp. Algorithms* **2013**, *4*, 1–16.
18. Swift, A.; Lindfield, G.R. Comparison of a continuation method with Brent's method for the numerical solution of a single nonlinear equation. *Comput. J.* **1978**, *21*, 359–362. [CrossRef]
19. Suhadolnik, A. Combined bracketing methods for solving nonlinear equations. *Appl. Math. Lett.* **2012**, *25*, 1755–1760. [CrossRef]
20. BenchmarkDotNet: Powerful. NET Library for Benchmarking. Available online: https://benchmarkdotnet.org/ (accessed on 10 April 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.