An Improvement to the Brent's Method

Zhengqiu Zhang zqzhang@cams.cma.gov.cn
Chinese Academy of Meteorological Sciences
Beijing, 100081,China

Abstract

This study presents an improvement to Brent's method by reconstruction. The Brent's method determines the next iteration interval from two subsections, whereas the new method determines the next iteration interval from three subsections constructed by four given points and thus can greatly reduce the iteration interval length.

The new method not only gets more readable but also converges faster. An experiment is made to investigate its performance. Results show that, after simplification, the computational efficiency can greatly be improved.

Keywords: Brent's Method, Simplification, Improvement.

1. INTRODUCTION

Brent's method, which is proposed by Brent (1973)[1] and introduced in many numerical books [2], is a root-finding algorithm with combining root bracketing, bisection and inverse quadratic interpolation, based on Dekker's method [3] to avoid the problem that it converges very slowly, in particular, when $|b_k - b_{k-1}|$ may be arbitrarily small, where k is the index for iterative steps.

Brent (1973) published an Algol 60 implementation. Netlib contains a Fortran translation of this implementation with slight modifications. The MATLAB function fzero also implements Brent's method [4], as does the PARI/GP method solve. Other implementations of the algorithm (in C++, C, and Fortran) can be found in the Numerical Recipes books [5]. However, due to the difficulty to understand this algorithm, this method was replaced by Ridders' method [6], as mentioned in some books [7].

2. IMPROVEMENT OF THE BRENT'S METHOD

2.1 Comparisons Between the Brent's Method and the Simplification

The original Brent's algorithm can be easily found on many websites such as Wikipedia (Brent's method)[8], for comparison it is reprinted here:

- **input** *a*, *b*, and a pointer to a subroutine for *f*
- calculate *f*(*a*)
- calculate *f*(*b*)
- **if** *f*(*a*) *f*(*b*) >= 0 **then** error-exit **end if**
- **if** |*f*(*a*)| < |*f*(*b*)| **then swap** (*a*,*b*) **end if**
- *c* := *a*
- **set** mflag
- **repeat until** *f*(*b or s*) = 0 **or** |*b* – *a*| is small enough *(convergence)*
    - **if** *f*(*a*) ≠ *f*(*c*) **and** *f*(*b*) ≠ *f*(*c*) **then**
        - calculate *s* *(inverse quadratic interpolation)*
    - **else**

        - calculate *s* *(secant rule)*
    - **end if**
    - **if** *(condition 1) s* is not between (3*a* + *b*)/4 and *b*
      or *(condition 2)* (mflag is set **and** |*s*−*b*| ≥ |*b*−*c*| / 2)
      or *(condition 3)* (mflag is cleared **and** |*s*−*b*| ≥ |*c*−*d*| / 2)
      or *(condition 4)* (mflag is set **and** |*b*−*c*| < |δ|)
      or *(condition 5)* (mflag is cleared **and** |*c*−*d*| < |δ|)
      then
        - *(bisection method)*
        - **set** mflag
    - **else**
        - **clear** mflag
    - **end if**
    - calculate *f*(*s*)
    - *d* := *c (d is asigned for the first time here, it won't be used above on the first iteration because mflag is set)*
    - *c* := *b*
    - **if** *f*(*a*) *f*(*s*) < 0 **then** *b*:= *s* **else** *a* := *s* **end if**
    - **if** |*f*(*a*)| < |*f*(*b*)| **then swap** (*a*,*b*) **end if**
- **end repeat**
- **output** *b or s (return the root)*

As seen from the above, the Brent's method is very complicated and difficult to understand. It is obvious that, the algorithm still preserves the idea that determines the next iteration interval from two subsections, in which one endpoint of previous interval is replaced with the new one calculated using inverse quadratic interpolation.

According to the computation, three points will be used for the next inverse quadratic interpolation, one is *b*, another is *s*, which will be one of the two endpoints of the next interval, and the other will be *b* or *a*. When it takes *a* = *s*, then *f*(a)=*f*(*c*), the Brent's method will use the secant rule algorithm.

To simplify the Brent's method, we can add one more evaluation for the function at the middle point *c* = (*a* + *b*)/2 before the interpolation. The simplified scheme now becomes as follows:

- **input** *a*, *b*, and a pointer to a subroutine for *f*
- calculate *f*(*a*)
- calculate *f*(*b*)
- **if** *f*(*a*) *f*(*b*) >= 0 **then** error-exit **end if**
- **repeat until** *f*(*b or s*) = 0 **or** |*b* − *a*| is small enough *(convergence)*
    - *c* =(*a*+*b*)/2
    - calculate *f*(*c*)
    - **if** *f*(*a*) ≠ *f*(*c*) **and** *f*(*b*) ≠ *f*(*c*) **then**
        - calculate *s* *(inverse quadratic interpolation)*
    - **else**
        - calculate *s* *(secant rule)*
    - **end if**
    - calculate *f*(*s*)
    - **if** *c*> *s* **then swap**(s,c)
    - **if** *f*(*c*) *f*(*s*) < 0 **then**
        - a: = s
        - b: = c
    - **else**
        - **if** *f*(*s*) *f*(*b*) < 0 **then** a: = c **else** b: = s **endif**
    - **end if**
- **end repeat**

- **output** *b or s (return the root)*

As seen from the above, after simplification, the new algorithm becomes much brief, more easily readable and understandable. Also, we don't need set the *mflag*, reducing several of its related conditions.

In the new algorithm, it is assumed that $a < b$, and c is the middle. If the interploated point (s) is less than c, then swap s and c, which makes the four points intersected with x–axis be labelled in the order of a, c, s and b by their coordinates.

Note that, for the above two methods, to avoid calulating the function again, when swapping or assigning a variable to another, the function variable also needs to do the same, accordingly.

2.2 Analysis on the Advantages of the New Algorithm

Because the new algorithm added one more evaluation for the root-finding function, one step of iteration for it will be equivalent to two steps of the Brent's method. This will provide it with several advantages over the original Brent's method.

a) Reducing the Times for Evaluating the Conditions

Since for each step the Brent's method must evaluate the condition: if ($f_1 \times f_2 < 0$), two iterations must need two times of such judgment. Whereas for the new algorithm, its conditional judgment is: If…else, two times of evaluating the root-finding function each loop just does the judgment once.

Therefore, it is obvious that the new algorithm spends less time to do the judgment after two times of the function evaluation than the Brent's method.

b) Accelerating Reduction of the Convergence Interval

By introducing the evaluation at the middle point, in each iterative loop for the new algorithm, the convergence interval must reduce more than half of the previous interval, but two iterations of the Brent's method could not.

Although it will get more accurate interpolated point using the inverse quadratic interpolation, two iterations can't guarantee the next interval reduces half of its previous. It is easy to find such an example. Suppose current iterative interval is [a, b], which can be divided into two half. If two continuous interpolated points are located in the same half region, it can't guarantee the last interval length reduces that much.

Since the Brent's method uses the bisection method under some conditions, it is obvious that the new method can more greatly reduce the next interval length.

2.3 Comparisons With Ridders' Method

Before modification, Ridder's method is simpler than Brent's method, but Press et al. claim that it usually performs about as well [9]. However, the new revised Brent's method becomes as simple as the Ridder's method. They have the same feature as that they converge quadratically, which implies that the number of additional significant digits doubles at each step, and the function has to be evaluated twice for each step, but the revised Brent's method converges faster.

The Ridders' method can be described as follows ("exponential case"). Given three x values $\{x_0, x_1, x_2\}$ evenly spaced by interval d_0 at which the function $f(x)$ has been calculated, a function $p(x)$ in an exponential form is found which takes the same values as $f(x)$ at the three points, then x_3 is found as the point where $p(x) = 0$. Once x_3 has been found, the closest one of the original three points is used as one of the three new points, along with one more new point chosen so as to have three equally spaced points with x_3 in the middle.

Compared with Ridders' method, although the simplified Brent's method and the Ridders' method both determine the next iterative subsection via four points $\{x_0, x_1, x_2, x_3\}$, the simplified Brent's method will have a narrower converging subsection for the next iteration than that of Ridders' method does, as can be seen from their actions to search for the converging subsection. That is, the next iterative subsection for the new method is one of the three subsections formed by the four points, whereas the next iterative subsection for Ridders' method is the subsection double extended on one of the three subsections.

3. EXPERIMENTAL TEST

From the above analysis, we can tell the new algorithm will be faster than the Brent's method. To confirm it, the following experiment will be used to test and compare their efficiencies. Simply, let's find one root of

$$\cos(x) - x^3 = 0,$$

Using the two methods with iteration, respectively. For the initial, set a=-4 and b=4 as the two endpoints of a section in which the root will be located. And iteration termination error is given to be 1.0E-5.

3.1 Analysis on the Advantages of the New Algorithm

Due to one more evaluation of the root-finding function is needed for the new method in each do-loop, equivalent to two times of iterations for the Brent's method. To make them comparable, we compare the times of the function evaluation for the two methods.

Step	Brent's method			The new method		
	(a, b)		$\lvert\Delta\rvert$	(a, b)		$\lvert\Delta\rvert$
1	4 .000000	0.000000	4.000000	—	—	—
2	2.000000	0.000000	2.000000	0.000000	4.000000	4.000000
3	0.000000	1.000000	1.000000	—	—	—
4	1.000000	0.685073	0.314927	0.235070	2.000000	1.764930
5	1.000000	0.842537	0.157463	—	—	—
6	0.921268	0.842537	0.078732	0.710220	1.117535	0.407315
7	0.842537	0.881903	0.039366	—	—	—
8	0.881903	0.865107	0.016796	0.862843	0.913877	0.051035
9	0.873505	0.865107	0.008398	—	—	—
10	0.869306	0.865107	0.004199	0.865470	0.888360	0.022890
11	0.867206	0.865107	0.002099	—	—	—
12	0.866156	0.865107	0.001050	0.865470	**0.865474**	0.000004
13	0.865107	0.865631	0.000525			
14	0.865631	0.865474	0.000157			
15	0.865553	0.865474	0.000079			
16	0.865513	0.865474	0.000039			
17	0.865494	0.865474	0.000020			

TABLE 1: Comparisons between the results from the two methods.

Before comparison, we design two programs for the two methods with C or other languages, respectively. The endpoints *a*, *b* and the interval length at each step are printed as in Table 1.

In the table, the "—" represents the function has to be evaluated twice. As seen from the table, under the given iteration termination error, the new method can reach the real root more quickly, with fewer iteration steps. Similar to Dekker's method (1969), when $|a-b|$ gets arbitrarily small, the computation of the Brent's method converges very slowly.

3.2 Converging Speed

Generally, there are several factors that affect the converging speed, which includes the calculations of the function $f(x)$, the interpolation, the conditional judgment and so on.

In practice, we can use computer system functions to record time consumptions for the both methods. However, for simplicity, the following just gives an analysis on their computational efficiencies.

As seen from the above experiment, when getting the root of the equation, the new method needs to calculate 12 times of the function:

$$f(x) = \cos(x) - x^3,$$

While the Brent's method needs to calculate $f(x)$ for 18 times. Since $f(x)$ is the same, indicating one time calculation of the function for the two methods need the same computational time, thus the new method spends less total time on evaluating the $f(x)$. On the other hand, the above analysis and the code's structures also indicated that the new method needs less time.

Briefly, comparing with the original Brent's method, the new algorithm not only gets more understandable but also will improve its computational efficiency.

4. CONCLUSIONS

This study proposes an improvement to the Brent's method, and a comparative experiment test was conducted. The new algorithm is simpler and more easily understandable. Experimental results and analysis indicated that the proposed method converges faster. Other experiments also show this advantage.

In short, the proposed method has a lot of advantages over the original Brent's method, and it will be useful for engineering computations.

5. REFERENCES

[1] Brent, R.P., Algorithms for Minimization without Derivatives, Chapter 4. Prentice- Hall, Englewood Cliffs, NJ. ISBN 0-13-022335-2,1973.

[2] Antia,H.M., Numerical Methods for Scientists and Engineers, Birkhäuser, 2002, pp.362-365,2 ed.

[3] Dekker, T. J. Finding a zero by means of successive linear interpolation, In B. Dejon and P. Henrici (eds), Constructive Aspects of the Fundamental Theorem of Algebra, Wiley-Interscience, London, SBN 471-28300-9,1969.

[4] Alfio Quarteroni, Fausto Saleri. Scientific Computing with MATLAB (Texts in Computational Science and Engineering 2), Springer, 2003, pp.52.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery. Numerical Recipes in C, The Art of Scientific Computing Second Edition, Cambridge University Press, November 27, 1992, pp. 358–362.

[6] Ridders, C.J.F. " Three-point iterations derived from exponential curve fitting ", IEEE Transactions on Circuits and Systems 26 (8): 669-670,1979.

[7] Jaan Kiusalaas. Numerical Methods in Engineering with Python, 2nd Edition, Cambridge University Press, 2010.

[8] Wikipedia contributors. " Brent's method. Wikipedia ", The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 13 Apr. 2010. Web. 13 May. 2010.

[9] Press, W.H.; S.A. Teukolsky, W.T. Vetterling, B.P. Flannery. Numerical Recipes in C: The Art of Scientific Computing (2nd ed.). Cambridge UK: Cambridge University Press.1992, pp. 358–359.