Brent's method

From Wikipedia, the free encyclopedia

In numerical analysis, **Brent's method** is a complicated but popular root-finding algorithm combining the bisection method, the secant method and inverse quadratic interpolation. It has the reliability of bisection but it can be as quick as some of the less reliable methods. The algorithm tries to use the potentially fast-converging secant method or inverse quadratic interpolation if possible, but it falls back to the more robust bisection method if necessary. Brent's method is due to Richard Brent[1] and builds on an earlier algorithm by Theodorus Dekker.[2] Consequently, the method is also known as **Brent-Dekker**.

Contents

Dekker's method

The idea to combine the bisection method with the secant method goes back to Dekker (1969).

Suppose that we want to solve the equation $f(x) = 0$. As with the bisection method, we need to initialize Dekker's method with two points, say a_0 and b_0, such that $f(a_0)$ and $f(b_0)$ have opposite signs. If f is continuous on $[a_0, b_0]$, the intermediate value theorem guarantees the existence of a solution between a_0 and b_0.

Three points are involved in every iteration:

- b_k is the current iterate, i.e., the current guess for the root of f.
- a_k is the "contrapoint," i.e., a point such that $f(a_k)$ and $f(b_k)$ have opposite signs, so the interval $[a_k, b_k]$ contains the solution. Furthermore, $|f(b_k)|$ should be less than or equal to $|f(a_k)|$, so that b_k is a better guess for the unknown solution than a_k.
- b_{k-1} is the previous iterate (for the first iteration, we set $b_{k-1} = a_0$).

Two provisional values for the next iterate are computed. The first one is given by linear interpolation, also known as the secant method:

$$s = \begin{cases} b_k - \frac{b_k - b_{k-1}}{f(b_k) - f(b_{k-1})} f(b_k), & \text{if } f(b_k) \neq f(b_{k-1}) \\ m & \text{otherwise} \end{cases}$$

and the second one is given by the bisection method

$m = \frac{a_k + b_k}{2}$. If the result of the secant method, s, lies strictly between b_k and m, then it becomes the next iterate ($b_{k+1} = s$), otherwise the midpoint is used ($b_{k+1} = m$).

Then, the value of the new contrapoint is chosen such that $f(a_{k+1})$ and $f(b_{k+1})$ have opposite signs. If $f(a_k)$ and $f(b_{k+1})$ have opposite signs, then the contrapoint remains the same: $a_{k+1} = a_k$. Otherwise, $f(b_{k+1})$ and $f(b_k)$ have opposite signs, so the new contrapoint becomes $a_{k+1} = b_k$.

Finally, if $|f(a_{k+1})| < |f(b_{k+1})|$, then a_{k+1} is probably a better guess for the solution than b_{k+1}, and hence the values of a_{k+1} and b_{k+1} are exchanged.

This ends the description of a single iteration of Dekker's method.

Dekker's method performs well if the function f is reasonably well-behaved. However, there are circumstances in which every iteration employs the secant method, but the iterates b_k converge very slowly (in particular, $|b_k - b_{k-1}|$ may be arbitrarily small). Dekker's method requires far more iterations than the bisection method in this case.

Brent (1973) proposed a small modification to avoid this problem. He inserted an additional test which must be satisfied before the result of the secant method is accepted as the next iterate. Two inequalities must be simultaneously satisfied:

- given a specific numerical tolerance δ,

```
            if the previous step used the bisection method, the inequality
```

  $|\delta| < |b_k - b_{k-1}|$ must hold to perform interpolation, otherwise the bisection method is performed and its result used for the next iteration.

```
        If the previous step performed interpolation, then the inequality
```

  $|\delta| < |b_{k-1} - b_{k-2}|$ is used instead to perform the next action (to choose) interpolation (when inequality is true) or bisection method

```
   (when inequality is not true).
```

- Also, if the previous step used the bisection method, the inequality

  $|s - b_k| < \frac{1}{2}|b_k - b_{k-1}|$ must hold, otherwise the bisection method is performed and its result used for the next iteration.

If the previous step performed interpolation, then the inequality

$|s - b_k| < \frac{1}{2}|b_{k-1} - b_{k-2}|$ is used instead.

This modification ensures that at the kth iteration, a bisection step will be performed in at most $2\log_2(|b_{k-1} - b_{k-2}|/\delta)$ additional iterations, because the above conditions force consecutive interpolation step sizes to halve every two iterations, and after at most $2\log_2(|b_{k-1} - b_{k-2}|/\delta)$ iterations, the step size will be smaller than δ, which invokes a bisection step. Brent proved that his method requires at most N^2 iterations, where N denotes the number of iterations for the bisection method. If the function f is well-behaved, then Brent's method will usually proceed by either inverse quadratic or linear interpolation, in which case it will converge superlinearly.

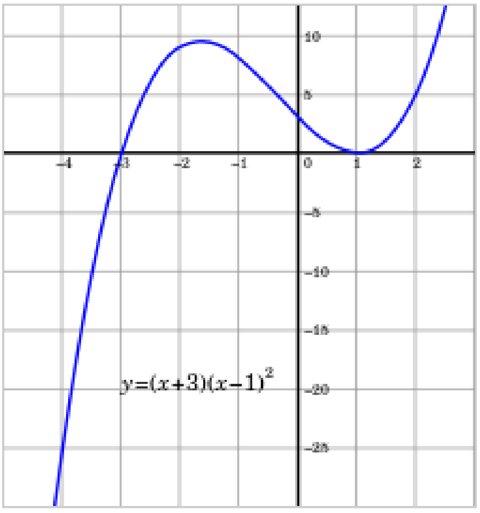
Furthermore, Brent's method uses inverse quadratic interpolation instead of linear interpolation (as used by the secant method) .If $f(b_k)$, $f(a_k)$ and $f(b_{k-1})$ are distinct, it slightly increases the efficiency. As a consequence, the condition for accepting s (the value proposed by either linear interpolation or inverse quadratic interpolation) has to be changed: s has to lie between $(3a_k + b_k) / 4$ and b_k.

Algorithm

```
input a, b, and (a pointer to) a function for f
calculate f(a)
calculate f(b)
if f(a) f(b) >= 0 then exit function because the root is not bracketed.
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
    $s := \frac{af(b)f(c)}{(f(a)-f(b))(f(a)-f(c))} + \frac{bf(a)f(c)}{(f(b)-f(a))(f(b)-f(c))} + \frac{cf(a)}{(f(c)-f(a))(}$
  else
    $s := b - f(b)\frac{b-a}{f(b)-f(a)}$ (secant method)
  end if
  if (condition 1) s is not between $\frac{3a+b}{4}$ 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
    $s := \frac{a+b}{2}$ (bisection method)
    set mflag
  else
    clear mflag
  end if
  calculate f(s)
  d := c  (d is assigned for the first time here; it won't be used above on the first iteration because mflag is
  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)
```

Example

Suppose that we are seeking a zero of the function defined by $f(x) = (x + 3)(x - 1)^2$. We take $[a_0, b_0] = [-4, 4/3]$ as our initial interval. We have $f(a_0) = -25$ and $f(b_0) = 0.48148$ (all numbers in this section are rounded), so the conditions $f(a_0)\,f(b_0) < 0$ and $|f(b_0)| \le |f(a_0)|$ are satisfied.



Graph of $f(x) = (x + 3)(x - 1)^2$

1. In the first iteration, we use linear interpolation between $(b_{-1}, f(b_{-1})) = (a_0, f(a_0)) = (-4, -25)$ and $(b_0, f(b_0)) = (1.33333, 0.48148)$, which yields $s = 1.23256$. This lies between $(3a_0 + b_0) / 4$ and b_0, so this value is accepted. Furthermore, $f(1.23256) = 0.22891$, so we set $a_1 = a_0$ and $b_1 = s = 1.23256$.
2. In the second iteration, we use inverse quadratic interpolation between $(a_1, f(a_1)) = (-4, -25)$ and $(b_0, f(b_0)) = (1.33333, 0.48148)$ and $(b_1, f(b_1)) = (1.23256, 0.22891)$. This yields 1.14205, which lies between $(3a_1 + b_1) / 4$ and b_1. Furthermore, the inequality $|1.14205 - b_1| \le |b_0 - b_{-1}| / 2$ is satisfied, so this value is accepted. Furthermore, $f(1.14205) = 0.083582$, so we set $a_2 = a_1$ and $b_2 = 1.14205$.
3. In the third iteration, we use inverse quadratic interpolation between $(a_2, f(a_2)) = (-4, -25)$ and $(b_1, f(b_1)) = (1.23256, 0.22891)$ and $(b_2, f(b_2)) = (1.14205, 0.083582)$. This yields 1.09032, which lies between $(3a_2 + b_2) / 4$ and b_2. But here Brent's additional condition kicks in: the inequality $|1.09032 - b_2| \le |b_1 - b_0| / 2$ is not satisfied, so this value is rejected. Instead, the midpoint $m = -1.42897$ of the interval $[a_2, b_2]$ is computed. We have $f(m) = 9.26891$, so we set $a_3 = a_2$ and $b_3 = -1.42897$.
4. In the fourth iteration, we use inverse quadratic interpolation between $(a_3, f(a_3)) = (-4, -25)$ and $(b_2, f(b_2)) = (1.14205, 0.083582)$ and $(b_3, f(b_3)) = (-1.42897, 9.26891)$. This yields 1.15448, which is not in the interval between $(3a_3 + b_3) / 4$ and b_3). Hence, it is replaced by the midpoint $m = -2.71449$. We have $f(m) = 3.93934$, so we set $a_4 = a_3$ and $b_4 = -2.71449$.
5. In the fifth iteration, inverse quadratic interpolation yields −3.45500, which lies in the required interval. However, the previous iteration was a bisection step, so the inequality $|-3.45500 - b_4| \le |b_4 - b_3| / 2$ need to be satisfied. This inequality is false, so we use the midpoint $m = -3.35724$. We have $f(m) = -6.78239$, so m becomes the new contrapoint ($a_5 = -3.35724$) and the iterate remains the same ($b_5 = b_4$).
6. In the sixth iteration, we cannot use inverse quadratic interpolation because $b_5 = b_4$. Hence, we use linear interpolation between $(a_5, f(a_5)) = (-3.35724, -6.78239)$ and $(b_5, f(b_5)) = (-2.71449, 3.93934)$. The result is $s = -2.95064$, which satisfies all the conditions. But since the iterate did not change in the previous step, we reject this result and fall back to bisection. We update $s = -3.03587$, and $f(s) = -0.58418$.
7. In the seventh iteration, we can again use inverse quadratic interpolation. The result is $s = -3.00219$, which satisfies all the conditions. Now, $f(s) = -0.03515$, so we set $a_7 = b_6$ and $b_7 = -3.00219$ (a_7 and b_7 are exchanged so that the condition $|f(b_7)| \le |f(a_7)|$ is satisfied). *(Correct : linear interpolation $s = -2.99436$, $f(s) = 0.089961$)*
8. In the eighth iteration, we cannot use inverse quadratic interpolation because $a_7 = b_6$. Linear interpolation yields $s = -2.99994$, which is accepted. *(Correct : $s = -2.9999$, $f(s) = 0.0016$)*
9. In the following iterations, the root $x = -3$ is approached rapidly: $b_9 = -3 + 6 \cdot 10^{-8}$ and $b_{10} = -3 -$

$3 \cdot 10^{-15}$. *(Correct : Iter 9 : f(s) = -1.4E-07, Iter 10 : f(s) = 6.96E-12)*

Implementations

- Brent (1973) published an Algol 60 implementation.
- Netlib contains a Fortran translation of this implementation with slight modifications.
- The PARI/GP method `solve` implements the method.
- Other implementations of the algorithm (in C++, C, and Fortran) can be found in the Numerical Recipes books.
- The Apache Commons Math library implements the algorithm in Java.
- The Scipy optimize module implements the algorithm in Python (programming language)
- The Modelica Standard Library implements the algorithm in Modelica.
- The `optimize` function implements the algorithm in R (software).
- The Boost (C++ libraries) implements the algorithm in C++ in the Math toolkit ("Locating function minima").

References

1. Brent 1973
2. Dekker 1969

- Brent, R. P. (1973), "Chapter 4: An Algorithm with Guaranteed Convergence for Finding a Zero of a Function", *Algorithms for Minimization without Derivatives*, Englewood Cliffs, NJ: Prentice-Hall, ISBN 0-13-022335-2
- Dekker, T. J. (1969), "Finding a zero by means of successive linear interpolation", in Dejon, B.; Henrici, P., *Constructive Aspects of the Fundamental Theorem of Algebra*, London: Wiley-Interscience, ISBN 978-0-471-20300-1

Further reading

- Atkinson, Kendall E. (1989). "Section 2.8.". *An Introduction to Numerical Analysis* (2nd ed.). John Wiley and Sons. ISBN 0-471-50023-2.
- Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P. (2007). "Section 9.3. Van Wijngaarden–Dekker–Brent Method". *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Zhang, Zhengqiu (2011), "An Improvement to the Brent's Method" (PDF), *International Journal of Experimental Algorithms* **2** (1): 21–26. This article claims to simplify and improve Brent's method.
- Stage, Steven A. (2013), "Comments on An Improvement to the Brent's Method" (PDF), *International Journal of Experimental Algorithms* **4** (1): 1–16. This article points out and corrects errors in Zhang's algorithm and then compares several methods (not just Zhang and Brent).

External links

- zeroin.f (http://www.netlib.org/go/zeroin.f) at Netlib.
- Module for Brent's Method (http://math.fullerton.edu/mathews/n2003/BrentMethodMod.html) by John H. Mathews
- module brent in C++ (also C, Fortran, Matlab)

(http://people.sc.fsu.edu/~jburkardt/cpp_src/brent/brent.html) by John Burkardt
- GSL (http://www.gnu.org/software/gsl/) implementation.
- Boost C++ (http://www.boost.org/doc/libs/1_55_0/libs/math/doc/html/math_toolkit/internals1/minima.html) implementation.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Brent%27s_method&oldid=681782808"

Categories: Root-finding algorithms
