The Quartile Algorithm for Root Bracketing

by

Namir Shammas

Introduction

Calculating the roots for functions and polynomials has gained the attention of mathematicians for many centuries. Students of numerical analysis come across a rich harvest of algorithms ranging from the simple to the complex. One class of root-seeking algorithms is called the root-bracketing algorithms in which the solution requires that you start with an interval that contains the root. The targeted function when evaluated at the ends of the interval give values that have opposite signs. The Bisection method is the simplest and least efficient root-bracketing algorithm.

This paper introduces a brand new root-bracketing algorithm designed by the author. The new Quartile algorithm is more efficient than the Bisection method. While the Bisection compares the signs of the function's values, the new algorithm compares the absolute function's value. This strategy allows it to narrow the root-bracketing interval faster than the Bisection.

I first presented this algorithm at the HHC2005 (Hand Held Conference) conference which took place in Chicago in the year 2005.

Algorithm

First, let me present the algorithm for the Bisection method since I am billing the Quartile algorithm as better than the Bisection.

To solve $f(X) = 0$ given interval $[A, B]$ such that $f(A) * f(B) <= 0$ and the Tolerance = maximum interval width that is acceptable for the solution:

```
1. $Fa = F(A)$ and $Fb = F(B)$
2. Calculate $M = (A + B) / 2$
3. $Fm = F(M)$
4. If $fm * fa > 0$
    a. Then $A = M$ and $Fa = Fm$
    b. Else $B = M$ and $Fb = Fm$
5. If $|A - B| >$ Tolerance Then go to step 2
6. Return root as $M$ or $(A + B) /2$
```

The algorithm for the Quartile method is:

To solve f(X) = 0 given interval [A, B] such that f(A) * f(B) <= 0, a coefficient α (in the range of 0.25 to 0.3), and the Tolerance = maximum interval width that is acceptable for the solution:

1. Fa = F(A) and Fb = F(B)
2. If |Fb| > |Fa|
    a. Then M = A + α * (B – A)
    b. Else M = B + α * (A – B)
3. Fm = F(M)
4. If fm * fa > 0
    a. Then A = M and Fa = Fm
    b. Else B = M and Fb = Fm
5. If |A - B| > Tolerance Then go to step 2
6. Return root as M or (A + B) /2

The coefficient α tells the algorithm how close M is calculated to either A or B based on comparing the absolute values of f(A) and f(B). When you set α to 0.5, the Quartile method becomes the Bisection method. Thus you can say that the Bisection method is a special case of the Quartile algorithm.

Comparison with Bisection

To compare the Quartile and Bisection methods, consider the following function:

$f(X) = \exp(X) - 3 * X^2$

The function has roots near -0.45, 0.91, and 3.73. The following table compares the number of function calls needed to reach a solution using a tolerance of 1e-7.

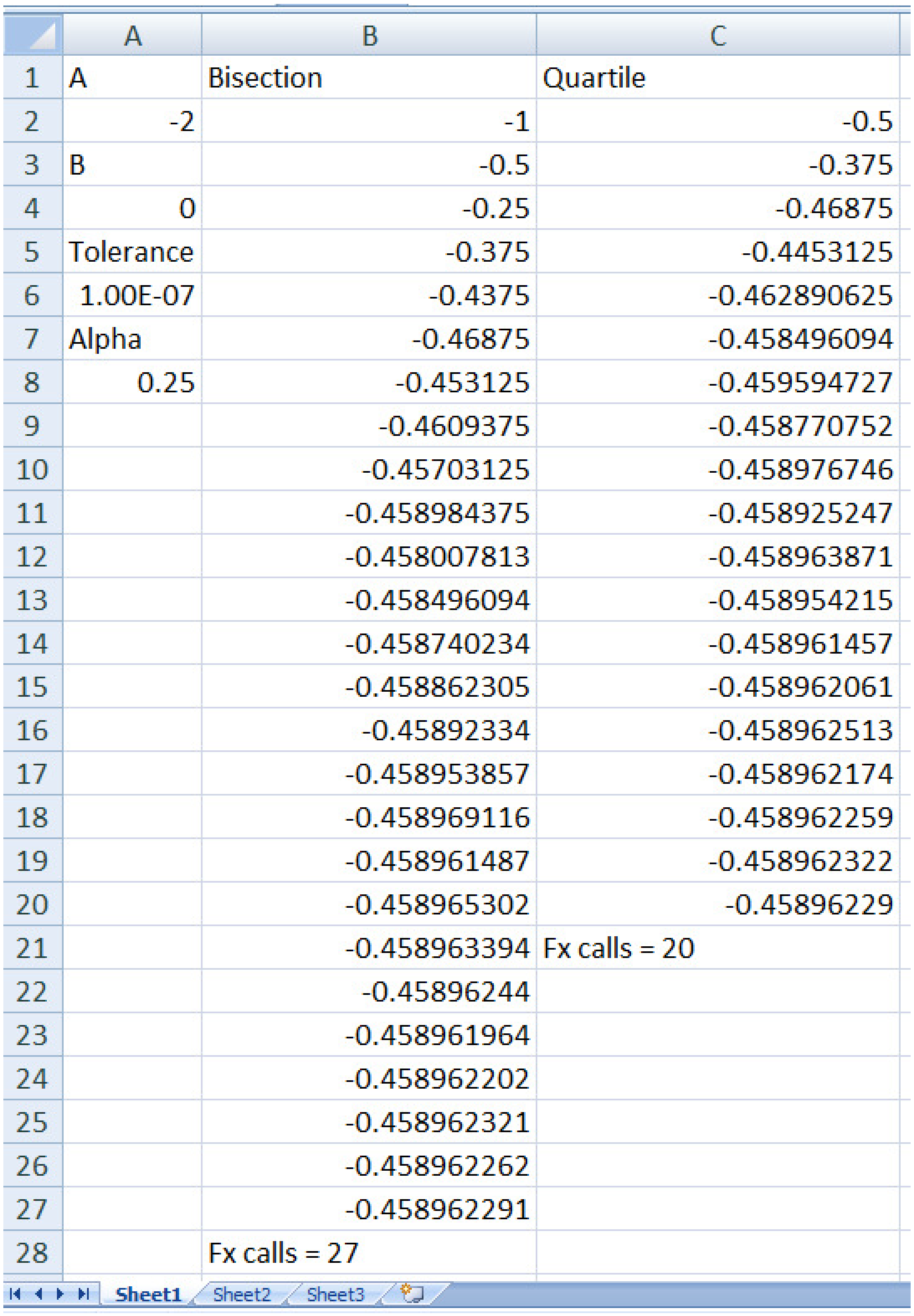
Initial A	*Initial B*	*Bisection fx calls*	*Quartile fx calls*
3	4	26	20
2	4	27	19
0	1	26	20
0	2	27	22

Initial A	***Initial B***	***Bisection fx calls***	***Quartile fx calls***
-1	0	26	21
-2	0	27	20

The table shows that in each case, the Quartile algorithms required fewer function calls than the Bisection to reach the refined estimate for the root.

The Test Excel Spreadsheet and VBA Code

The following Excel spreadsheet was used to obtain the above results:



	A	B	C
1	A	Bisection	Quartile
2	-2	-1	-0.5
3	B	-0.5	-0.375
4	0	-0.25	-0.46875
5	Tolerance	-0.375	-0.4453125
6	1.00E-07	-0.4375	-0.462890625
7	Alpha	-0.46875	-0.458496094
8	0.25	-0.453125	-0.459594727
9		-0.4609375	-0.458770752
10		-0.45703125	-0.458976746
11		-0.458984375	-0.458925247
12		-0.458007813	-0.458963871
13		-0.458496094	-0.458954215
14		-0.458740234	-0.458961457
15		-0.458862305	-0.458962061
16		-0.45892334	-0.458962513
17		-0.458953857	-0.458962174
18		-0.458969116	-0.458962259
19		-0.458961487	-0.458962322
20		-0.458965302	-0.45896229
21		-0.458963394	Fx calls = 20
22		-0.45896244	
23		-0.458961964	
24		-0.458962202	
25		-0.458962321	
26		-0.458962262	
27		-0.458962291	
28		Fx calls = 27	

The data in column A must be entered in order to prepare for the test. The output (including the headings) appears in columns B and FxCallCounter.

The following VBA code was used for the comparative test:

```
Option Explicit

Function F(ByVal X As Double) As Double
  F = Exp(X) - 3 * X ^ 2
End Function

Sub CalcRoot()
  Dim A As Double, B As Double, M As Double
  Dim Fa As Double, Fb As Double, Fm As Double
  Dim Alpha As Double, Toler As Double
  Dim R As Integer, FxCallCounter As Integer

  A = Cells(2, 1)
  B = Cells(4, 1)
  Toler = Cells(6, 1)
  Alpha = Cells(8, 1)
  Range("B:C").Clear
  Range("B1").Value = "Bisection"


  ' Bisection method
  Fa = F(A)
  Fb = F(B)
  FxCallCounter = 2
  R = 2
  Do
    M = (A + B) / 2
    Fm = F(M)
    FxCallCounter = FxCallCounter + 1
    If Fm * Fa > 0 Then
      A = M
      Fa = Fm
    Else
      B = M
      Fb = Fm
    End If
    Range("B" & R).Value = M
    R = R + 1
  Loop Until Abs(A - B) <= Toler
  Range("B" & R).Value = (A + B) / 2
  Range("B" & CStr(R + 1)).Value = "Fx calls = " &
FxCallCounter

  A = Cells(2, 1)
  B = Cells(4, 1)
  Range("C1").Value = "Quartile"
  ' Quartile method
  Fa = F(A)
  Fb = F(B)
  FxCallCounter = 2
  R = 2
  Do
    If Abs(Fb) > Abs(Fa) Then
      M = A + Alpha * (B - A)
    Else
      M = B + Alpha * (A - B)
    End If
    Fm = F(M)
    FxCallCounter = FxCallCounter + 1
    If Fm * Fa > 0 Then
      A = M
      Fa = Fm
    Else
      B = M
      Fb = Fm
    End If
    Range("C" & R).Value = M
    R = R + 1
  Loop Until Abs(A - B) <= Toler
  Range("C" & R).Value = (A + B) / 2
  Range("C" & CStr(R + 1)).Value = "Fx calls = " &
FxCallCounter

End Sub
```