DATABASE

Enhancing Newton's Method

By Namir Clement Shammas, June 01, 2002

0206: Algorithm Alley

Namir is a technical writer specializing in application and SDK documentation. He can be reached at nshammas@aol.com.

Many numerical analysis problems involve solving for the roots of nonlinear equations. Numerical analysis offers various kinds of algorithms to perform this task. In this article, I examine the Newton-Raphson method and show how you can enhance it.

Solving for the root of a single nonlinear function is affected by the curvature, maxima, minima, and oscillation of that function. In the case of functions that are not smooth, you are better off using root-bracketing methods like the Bisection or False Position. These methods start with a range of values that contains the root and systematically narrow that range around the root value. While these methods are robust in yielding an answer, there is a cost involved. Typically, root-bracketing methods require more iterations than other methods, such as the Newton-Raphson method (which I'll simply call "Newton's method"). In the case of smooth functions, you can use algorithms that offer faster convergence, such as Newton's method. The source code that implements Newton's method, along with executables and output text files, is available electronically; see "Resource Center," page 5.

Back to Basics

Newton's method is based on the Taylor expansion, see Example 1(a), where *r*, *x*, *f(x)*, *f'(x)*, and *f''(x)* are the root, the guess for the root, the function, the function's first derivative, and the function's second derivative, respectively. Newton's method uses the first two terms on the right side of the Taylor expansion to yield Example 1(b), which has proven to be an efficient and popular root-seeking algorithm. The weakness of Newton's method appears when the slope of the function near the root and/or its guess is very small. This kind of slope value causes the next guess for the root to shoot far off from the vicinity of the actual root value.

It is often easier to calculate the first derivative numerically using the approximation in Example 1(c), where *h* is a small increment value. Combining Examples 1(b) and 1(c) yields Example 1(d), which requires evaluating the function at three values; namely, the guess and two neighboring values near the guess.

Of course, you can estimate the value of the function's second derivative using Example 1(e), which indicates that estimating the second derivative requires evaluating the function at three values; namely, the guess and two neighboring values near the guess. With this knowledge, you can go back to Example 1(a) and include the third term that contains the second derivative, since there are no extra function calls involved to estimate the second derivative. Since *f(r)* is zero, you can rewrite Example 1(a) as Example 1(f). Including the second derivative in the root-seeking algorithm lets that algorithm take the function's curvature (offered by the value of the second derivative) into account. Example 1(f) is essentially a quadratic equation for the factor *(r-x)*. You can solve the quadratic equation and write the result as Example 1(g).

A Second Approach

Another way to solve Example 1(g) yields Example 2, which shows the root *r* on both sides of the equation. To use this equation, employ Example 1(d) to calculate an estimate for the value for *r* on the right-hand side of the equation.

Why employ Examples 1(g) and 2, you ask? When I first came up with Example 1(g) (which I call the "Quad Newton method") and implemented it in a prototype application, the algorithm gave mixed results. On one hand, it offered faster convergence than Newton's method. On the other hand, the discriminate value was often negative, yielding run-time errors. By contrast, Example 2 (which I will call the "Enhanced Newton method") is more stable despite the fact that it requires estimating the value for *r* using Example 1(d). I fine-tuned the algorithm by using the steps in Figure 1, which shows the pseudocode that involves calculating intermediate values for *r*. The pseudocode shows that Example 2 is applied twice after applying Example 1(d). The algorithm does not make additional calls to the function beyond *f(x)*, *f(x+h)*, and *f(x-h)*. The implementation for the algorithm stores the values returned by these function calls and then repeatedly recalls these values when needed.

The Test Code

Root.cpp (available electronically) contains the implementation of the two versions of the Enhanced Newton method. The listing declares a set of five math functions, the class *Root*, the function *main*, and a few auxiliary functions.

The class *Root* declares:

- The data member *m_nMaxIter* stores the maximum number of iterations.
- The data member *m_nIter* stores the number of iterations.
- The data member *m_fToler* stores the tolerance used to determine convergence.
- The constructor that initializes the data members.

2014
Dr. Dobb's Journal
- November - **Mobile Development**
- August - **Web Development**
- May - **Testing**
- February - **Languages**

Dr. Dobb's Tech Digest
- DevOps
- Open Source
- Windows and .NET programming
- The Design of Messaging Middleware and 10 Tips from Tech Writers
- Parallel Array Operations in Java 8 and Android on x86: Java Native Interface and the Android Native Development Kit

2013
- January - **Mobile Development**
- February - **Parallel Programming**
- March - **Windows Programming**
- April - **Programming Languages**
- May - **Web Development**
- June - **Database Development**
- July - **Testing**
- August - **Debugging and Defect Management**
- September - **Version Control**
- October - **DevOps**
- November- **Really Big Data**
- December - **Design**

2012
- January - **C & C++**
- February - **Parallel Programming**
- March - **Microsoft Technologies**
- April - **Mobile Development**
- May - **Database Programming**
- June - **Web Development**
- July - **Security**
- August - **ALM & Development Tools**
- September - **Cloud & Web Development**
- October - **JVM Languages**
- November - **Testing**
- December - **DevOps**

- The member function *getIters* returns the number of iterations.
- The member function *Newton* implements Newton's algorithm.
- The member function *ExRoot* implements the Enhanced Newton method (Example 2).
- The member function *QuadRoot* implements the Quad Newton method; Example 1(g). This member function returns 1.0e+100 if the value of the discriminate is negative.

The function *main* creates *objRoot* as an instance of class *Root*. The function tests the various math functions and uses the auxiliary function output to obtain the roots and generate output. The function output writes results to the console, the text file root.dat (which closely echoes the console output), and the comma-delimited text file rootCDD.dat.

The Results

Table 1 shows the results of finding one of the roots of the equation $exp(x)-3*x^2$ (plotted in Figure 2). This function has three roots near -0.45, 0.91, and 3.73. The table shows that the Enhanced Newton method (using Example 2) was faster than the traditional Newton method. The Quad Newton algorithm, Example 1(g), also outperformed Newton's method and, in most cases, was up to par with the Enhanced Newton method. Using the improved algorithms pays off in this case because of the significant curvature in the range of the root and guesses.

Table 2 shows the results of finding one of the roots of the equation *cos(x)-x* (plotted in Figure 3). The table shows that the Enhanced Newton method takes one or two iterations less than Newton's method to obtain the root. The table also shows that the Quad Newton algorithm failed in finding the root of the function. The function *cos(x)-x* has a smooth curvature, giving a little advantage for the Enhanced Newton method over Newton's method.

Table 3 shows the results related to solving for the root of function $(x+15)*(x+10)*(x+20)*(x-4.5)$. Figure 4 shows a plot for this function. The initial guesses direct the methods to zoom in on the root at 4.5. All the methods work well. The Enhanced Newton and Quad Newton methods are consistently one iteration ahead of Newton's method. While the function has multiple roots and inflection points, its curvature is smooth near the root at 4.5. This smooth curvature gives the improved algorithms a modest advantage.

Table 4 contains the results for the root of the function $cosh(x)^2+sinh(x)^2+2*x-11$. Figure 5 shows a plot of this function. Again, all of the methods work well. The Enhanced Newton and Quad Newton methods are significantly ahead of Newton's method for the initial guesses of 0.5 and 1, respectively. As the initial guesses increase, the new methods are one iteration ahead of Newton's method.

Table 5 holds the results for solving the root of function $100-x-x^2/2-x^3/3-x^4/4$. Figure 6 shows a plot of this function. The Enhanced Newton method and the classical Newton method zoom in on the root at 4.03105, while the Quad Newton method zooms in on the root at -4.77164. The Enhanced Newton method leads significantly over Newton's method because of the function's significant curvature near the root.

Conclusion

The Enhanced Newton and Quad Newton methods offer faster root-seeking methods than the classical Newton method. The advantage varies depending on the mathematical function involved and the initial guesses. Remember that each iteration makes three calls to the mathematical function. When the mathematical function is complex (such as one that involves summations), the saving is relevant. This article presented various types of mathematical functions that showed how the three methods behaved. In all cases, the Enhanced Newton and Newton's method converged to a root. I recommend that you plot the curves for the function (or family of functions) and examine the slope and curvature near the initial guess and root. This kind of graph gives you a good idea of how much advantage you get with the Enhanced Newton method.

DDJ

1 2 3 4 5 6 7 8 9 10 Next

Related Reading

- News
- Commentary

- **Tools To Build Payment-Enabled Mobile Apps**
- **Restlet Completes "Complete" API Platform**
- **A Datacenter Operating System For Data Developers**
- **Sencha Licks Android 5.0 Lollipop, And Likes**
  More News»

- Slideshow
- Video

- **Jolt Awards 2015: Coding Tools**
- **Jolt Awards: The Best Books**
- **Developer Reading List**
- **Jolt Awards: The Best Testing Tools**
  More Slideshows»

- Most Popular

- **RESTful Web Services: A Tutorial**
- **Lambda Expressions in Java 8**
- **Developer Reading List: The Must-Have Books for JavaScript**
- **Why Build Your Java Projects with Gradle Rather than Ant or Maven?**
  More Popular»

More Insights

White Papers

- Privacy Management Solution Buyers Guide
- How a Trading Floor Continues its Operations During COVID-19 Lockdown

More >>

Reports

- Intelligent Outcomes Are Key to Business Resilience
- The State of DevSecOps Report

More >>

Webcasts

- Drafting a Data Breach Response Playbook
- Security Alert Fatigue: Tips for Taking Control

More >>

INFO-LINK

Login or Register to Comment