On the Order of Convergence of Iterative Methods

Chuck Allison
CNS 3320 – Numerical Software Engineering
Utah Valley State College

February 2006

Abstract

In finding the root of a non-linear equation in a single variable by an iterative method it is desirable to obtain maximum efficiency. That efficiency is measured by *order of convergence*, which this note explains.

Iterative Methods

Iterative methods for solving a non-linear equation, $f(x) = 0$, rewrite the equation as $x = g(x)$, and then use the latter formulation in an iterative manner: $x_{n+1} = g(x_n)$. The iterative process is said to converge when successive approximations are "sufficiently close". The best measure of closeness that can be expected is when the relative error between approximations reaches machine epsilon:

$$\left|\frac{x_{n+1} - x_n}{x_n}\right| \leq \epsilon$$

Since the spacing near x_n is no more than $\epsilon|x_n|$, the stopping criterion can also be interpreted as when the distance between successive approximations is itself no larger than the spacing between floating-point numbers in the neighborhood of the approximations:

$$|x_{n+1} - x_n| \leq \epsilon|x_n|$$

A sufficient condition for convergence is easily expressed in terms of the first derivative of $g(x)$ using the mean value theorem from elementary calculus, which states that for some z between a and b,

$$g(b) - g(a) = g'(z)(b - a)$$

Now consider the expression $|x_{n+1} - x_n|$, where the x values are iterates in the formula $x_{n+1} = g(x_n)$. We have

$$|x_{n+1} - x_n| = |g(x_n) - g(x_{n-1})|$$

but by the mean value theorem, for some z between x_{n+1} and x_n

$$|x_{n+1} - x_n| = |g(x_n) - g(x_{n-1})| = |g'(z)(x_n - x_{n-1})|$$

Suppose that g' is bounded in magnitude by some number, M, in an interval large enough to contain all the x_i. Then the following development is obtained:

$$\begin{aligned} |x_{n+1} - x_n| &\leq M|(x_n - x_{n-1})| \\ &= M|g(x_{n-1}) - g(x_{n-2})| \\ &\leq M^2|x_{n-1} - x_{n-2}| \\ &= M^2|g(x_{n-2}) - g(x_{n-3})| \\ &\leq M^3|x_{n-2} - x_{n-3}| \\ \cdots & \\ &\leq M^n|x_1 - x_0| \end{aligned}$$

A sufficient condition for convergence, therefore, is $M < 1$, or rather, $|g'(x)| < 1$ for all x in the interval of interest.

Example 1

Consider the quadratic equation $x^2 - 5x + 4$, with roots 4 and 1. If we rewrite this as $x = \frac{x^2+4}{5}$, and solve the inequality $|g'(x)| = |\frac{2x}{5}| < 1$, we find that convergence will be guaranteed if there is a root in the interval $-\frac{5}{2} < x < \frac{5}{2}$. The iterative process will converge to the root 1, then, if we choose x_0 in that range. The root 4 will not be found by this particular formulation.

Of utmost interest, though, is *how fast* the process converges. Executing the following program requires 41 iterations to converge to the root 1 with an initial guess of $x_0 = 2$.

```
// iterate.cpp: Root finding by fixed point iteration
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <limits>
#include "../ulps.h"
#include "../ieee.h"
using namespace std;

double g(double x) {
   return (x*x + 4) / 5.0;
}

int main(int argc, char* argv[]) {
   if (argc > 1) {
      double guess = atoi(argv[1]);
      double eps = numeric_limits<double>::epsilon();
      cout.precision(numeric_limits<double>::digits10+2);
      for (int i = 0; i < 50; ++i) {
         double next = g(guess);
         cout << i << ": " << next << " (" << ulps(next,guess) << ")\n";
         if (abs(next - guess) <= abs(guess)*eps)
            break;
         guess = next;
      }
   }
}
```

But Newton's method converges in just 6 iterations:

```
// newton.cpp: Root finding by Newton's Method
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <limits>
#include "../ieee.h"
#include "../ulps.h"
using namespace std;

double f(double x) {
    return x*x - 5*x + 4.0;
}

double df(double x) {
    return 2.0*x - 5.0;
}

double g(double x) {
   return x - f(x)/df(x);  // Newton's formula
}

int main(int argc, char* argv[]) {
   if (argc > 1) {
      double guess = atof(argv[1]);
      double eps = numeric_limits<double>::epsilon();
      double small = numeric_limits<double>::denorm_min();
      cout.precision(numeric_limits<double>::digits10+2);

      for (int i = 0; i < 50; ++i) {
         double next = g(guess);
         cout << i << ": " << next << " (" << ulps(next,guess) << ")\n";

         // Check for near zero just in case
         double fval = abs(f(next));
         if (fval <= small) {
            cout << "near zero (" << fval << ")\n";
            break;
         }

         // Check for FP numbers close enough (1-2 ulps)
         if (abs(next - guess) <= abs(guess)*eps)
            break;
         guess = next;
      }
   }
}
```

Why such a pronounced difference? The answer has to do with the *order of convergence* of the algorithms.

Order of Convergence

The order of convergence of an iterative algorithm describes the rate at which successive iterates become close to each other. Suppose we denote by e_i the error of the ith iterate from the true root of an equation: $e_i = x_i - r$. To use this quantity, we will first expand the iteration function, $g(x)$, in a Taylor series about the root, r,:

$$g(x) = g(r) + g'(r)(x - r) + g''(r)\frac{(x - r)^2}{2!} \dots$$

Now substitute $x = x_i$ and subtract $g(r)$ from both sides:

$$g(x_i) - g(r) = g'(r)(x_i - r) + g''(r)\frac{(x_i - r)^2}{2!} = e_i \dots$$

Remember that $x_{i+1} = g(x_i)$, and note that since the equation $x = g(x)$ should be satisfied exactly at the root, then $r = g(r)$, so we obtain

$$g(x_i) - g(r) = x_{i+1} - r = e_{i+1} = g'(r)(x_i - r) + g''(r)\frac{(x_i - r)^2}{2!} = g'(r)e_i + g''(r)\frac{e_i^2}{2!} \dots$$

We have now expressed the error of the $(i + 1)$th iteration (e_{i+1}) in terms of the error of the ith iteration (e_i). This gives us the full picture of the role the derivatives of the iteration function g play in the rate of convergence of iterative algorithms. The leading term, being the most significant, will have the greatest effect. The reason that Newton's method is so efficient is because the first derivative of g is zero at the root, r. To see this, recall that Newton's formula is $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$. This means that for Newton's method, $g(x) = x - \frac{f(x)}{f'(x)}$. Computing $g'(r)$ yields:

$$g'(r) = 1 - \frac{[f'(r)]^2 - f(r)f''(r)}{[f'(r)]^2} = \frac{f(r)f''(r)}{[f'(r)]^2} = 0 \tag{1}$$

This is zero because $f(r) = 0$ by definition. So when $f'(r) \neq 0$, $g'(r)$ vanishes, and the error of the $(i + 1)$th iterate varies as the square of the previous error. For this reason we say that Newton's method is a *second-order* (or *quadratic*) method. Simple, fixed-point iteration as seen in the beginning of this paper is *first-order* (or *linear*) since g' is not necessarily zero in general.

References

[1] P. Stark. *Introduction to Numerical Methods*. Macmillan, 1970.