3.2.4. Some General Philosophy of Root Finding

There are numerous methods for finding the roots of a nonlinear equation. The roots have specific values, and the method used to find the roots does not affect the values of the roots. However, the method can determine whether or not the roots can be found and the amount of work required to find them. Some general philosophy of root finding is presented below.

1. Bounding methods should bracket a root, if possible.
2. Good initial approximations are extremely important.
3. Closed domain methods are more robust than open domain methods because they keep the root bracketed in a closed interval.
4. Open domain methods, when they converge, generally converge faster than closed domain methods.
5. For smoothly varying functions, most algorithms will always converge if the initial approximation is close enough. The rate of convergence of most algorithms can be determined in advance.
6. Many, if not most, problems in engineering and science are well behaved and straightforward. In such cases, a straightforward open domain method, such as Newton's method presented in Section 3.4.2 or the secant method presented in Section 3.4.3, can be applied without worrying about special cases and peculiar behavior. If problems arise during the solution, then the peculiarities of the nonlinear equation and the choice of solution method can be reevaluated.
7. When a problem is to be solved only once or a few times, the efficiency of the method is not of major concern. However, when a problem is to be solved many times, efficiency of the method is of major concern.
8. Polynomials can be solved by any of the methods for solving nonlinear equations. However, the special techniques applicable to polynomials should be considered.
9. If a nonlinear equation has complex roots, that must be anticipated when choosing a method.
10. Analyst's time versus computer time must be considered when selecting a method.
11. Blanket generalizations about root-finding methods are generally not possible.

Root-finding algorithms should contain the following features:

1. An upper limit on the number of iterations.
2. If the method uses the derivative $f'(x)$, it should be monitored to ensure that it does not approach zero.
3. A convergence test for the change in the magnitude of the solution, $|x_{i+1} - x_i|$, or the magnitude of the nonlinear function, $|f(x_{i+1})|$, must be included.
4. When convergence is indicated, the final root estimate should be inserted into the nonlinear function $f(x)$ to guarantee that $f(x) = 0$ within the desired tolerance.

3.3 CLOSED DOMAIN (BRACKETING) METHODS

Two of the simplest methods for finding the roots of a nonlinear equation are:

1. Interval halving (bisection)
2. False position (regula falsi)

Except in linear problems, root finding invariably proceeds by iteration, and this is equally true in one or in many dimensions. Starting from some approximate trial solution, a useful algorithm will improve the solution until some predetermined convergence criterion is satisfied. For smoothly varying functions, good algorithms will always converge, *provided* that the initial guess is good enough. Indeed one can even determine in advance the rate of convergence of most algorithms.

It cannot be overemphasized, however, how crucially success depends on having a good first guess for the solution, especially for multidimensional problems. This crucial beginning usually depends on analysis rather than numerics. Carefully crafted initial estimates reward you not only with reduced computational effort, but also with understanding and increased self-esteem. Hamming's motto, "the purpose of computing is insight, not numbers," is particularly apt in the area of finding roots. You should repeat this motto aloud whenever your program converges, with sixteen-digit accuracy, to the wrong root of a problem, or whenever it fails to converge because there is actually *no* root, or because there is a root but your initial estimate was not sufficiently close to it.

"This talk of insight is all very well, but what do I actually do?" For one-dimensional root finding, it is possible to give some straightforward answers: You should try to get some idea of what your function looks like before trying to find its roots. If you need to mass-produce roots for many different functions, then you should at least know what some typical members of the ensemble look like. Next, you should always bracket a root, that is, know that the function changes sign in an identified interval, before trying to converge to the root's value.

Finally (this is advice with which some daring souls might disagree, but we give it nonetheless) never let your iteration method get outside of the best bracketing bounds obtained at any stage. We will see below that some pedagogically important algorithms, such as the *secant method* or *Newton-Raphson*, can violate this last constraint and are thus not recommended unless certain fixups are implemented.

Multiple roots, or very close roots, are a real problem, especially if the multiplicity is an even number. In that case, there may be no readily apparent sign change in the function, so the notion of bracketing a root — and maintaining the bracket — becomes difficult. We are hard-liners: We nevertheless insist on bracketing a root, even if it takes the minimum-searching techniques of Chapter 10 to determine whether a tantalizing dip in the function really does cross zero. (You can easily modify the simple golden section routine of §10.2 to return early if it detects a sign change in the function. And, if the minimum of the function is exactly zero, then you have found a *double* root.)

As usual, we want to discourage you from using routines as black boxes without understanding them. However, as a guide to beginners, here are some reasonable starting points:

- Brent's algorithm in §9.3 is the method of choice to find a bracketed root of a general one-dimensional function, when you cannot easily compute the function's derivative. Ridders' method (§9.2) is concise, and a close competitor.
- When you can compute the function's derivative, the routine `rtsafe` in §9.4, which combines the Newton-Raphson method with some bookkeeping on the bounds, is recommended. Again, you must first bracket your root. If you can easily compute *two* derivatives, then Halley's method (§9.4.2) is often worth a try.