CHAPTER 8

Nonlinear Systems

CONTENTS

TRY as we might, it is not possible to express all systems of equations in the linear framework we have developed over the last several chapters. Logarithms, exponentials, trigonometric functions, absolute values, polynomials, and so on are commonplace in practical problems, but none of these functions is linear. When these functions appear, we must employ a more general—but often less efficient—toolbox for nonlinear problems.

8.1 ROOT-FINDING IN A SINGLE VARIABLE

We begin by considering methods for root-finding in a single scalar variable. Given a function $f(x) : \mathbb{R} \to \mathbb{R}$, we wish to develop algorithms for finding points $x^* \in \mathbb{R}$ subject to $f(x^*) = 0$; we call x^* a *root* or *zero* of f. Single-variable problems in linear algebra are not particularly interesting; after all we can solve the equation $ax - b = 0$ in closed form as $x^* = b/a$. Roots of a nonlinear equation like $y^2 + e^{\cos y} - 3 = 0$, however, are less easily calculated.

8.1.1 Characterizing Problems

We no longer assume f is linear, but without *any* information about its structure, we are unlikely to make headway on finding its roots. For instance, root-finding is guaranteed to fail on

$$f(x) = \begin{cases} -1 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

or even more deviously (recall $\mathbb{Q}$ denotes the set of rational numbers):

$$f(x) = \begin{cases} -1 & x \in \mathbb{Q} \\ 1 & \text{otherwise.} \end{cases}$$

These examples are trivial in the sense that any reasonable client of root-finding software would be unlikely to expect it to succeed in this case, but more subtle examples are not much more difficult to construct.

For this reason, we must add some "regularizing" assumptions about f to make the root-finding problem well-posed. Typical assumptions include the following:

- *Continuity:* A function f is *continuous* if it can be drawn without lifting up a pen; more formally, f is continuous if the difference $f(x) - f(y)$ vanishes as $x \to y$.
- *Lipschitz:* A function f is *Lipschitz continuous* if there exists a constant c such that $|f(x) - f(y)| \leq c|x - y|$; Lipschitz functions need not be differentiable but are limited in their rates of change.
- *Differentiability:* A function f is *differentiable* if its derivative f' exists for all x.
- C^k*:* A function is C^k if it is differentiable k times and each of those k derivatives is continuous; C^∞ indicates that all derivatives of f exist and are continuous.

Example 8.1 (Classifying functions)**.** The function $f(x) = \cos x$ is C^∞ and Lipschitz on $\mathbb{R}$. The function $g(x) = x^2$ as a function on $\mathbb{R}$ is C^∞ but *not* Lipschitz. In particular, $|g(x) - g(0)| = x^2$, which cannot be bounded by any linear function of x as $x \to \infty$. When restricted to the unit interval $[0, 1]$, however, $g(x) = x^2$ can be considered Lipschitz since its slope is bounded by 2 in this interval; we say f is "locally Lipschitz" since this property holds on any interval $[a, b]$. The function $h(x) = |x|$ is continuous—or C^0—and Lipschitz but not differentiable thanks to its singularity at $x = 0$.

When our assumptions about f are stronger, we can design more effective algorithms to solve $f(x^*) = 0$. We will illustrate the spectrum trading off between generality and efficiency by considering a few algorithms below.

8.1.2 Continuity and Bisection

Suppose that all we know about f is that it is continuous. This is enough to state an intuitive theorem from single-variable calculus:

Theorem 8.1 (Intermediate Value Theorem)**.** Suppose that $f : [a, b] \to \mathbb{R}$ is continuous and that $f(a) < u < f(b)$ or $f(b) < u < f(a)$. Then, there exists $z \in (a, b)$ such that $f(z) = u$.

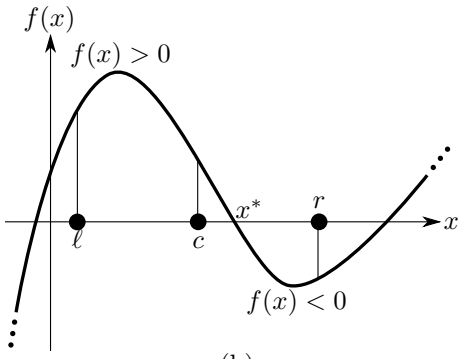
In other words, in the space between a and b, the function f must achieve every value between $f(a)$ and $f(b)$.

Suppose we are given as input a continuous function $f(x)$ as well as two values ℓ and r such that $f(\ell) \cdot f(r) < 0$; that is, $f(\ell)$ and $f(r)$ have opposite sign. Then, by the Intermediate Value Theorem, somewhere between ℓ and r there is a root of f. Similar to binary search, this property suggests a bisection algorithm for finding x^*, shown in Figure 8.1. This algorithm divides the interval $[\ell, r]$ in half recursively, each time keeping the side in which a root is known to exist by the Intermediate Value Theorem. It converges *unconditionally*, in the sense that ℓ and r are guaranteed to become arbitrarily close to one another and converge to a root x^* of $f(x)$.

Bisection is the simplest but not necessarily the fastest technique for root-finding. As with eigenvalue methods, bisection inherently is iterative and may never provide an *exact* solution x^*; this property is true for nearly any root-finding algorithm unless we put strong assumptions on the class of f. We can ask, however, how close the value c_k of the center

```
function BISECTION(f(x), ℓ, r)
  for k ← 1, 2, 3, ...
    c ← (ℓ+r)/2
    if |f(c)| < ε_f or |r − ℓ| < ε_x then
      return x* ≈ c
    else if f(ℓ) · f(c) < 0 then
      r ← c
    else
      ℓ ← c
```

(a)



(b)

Figure 8.1 (a) Pseudocode and (b) an illustration of the bisection algorithm for finding roots of continuous $f(x)$ given endpoints $\ell, r \in \mathbb{R}$ with $f(\ell) \cdot f(r) < 0$. The interval $[c, r]$ contains a root x^* because $f(c)$ and $f(r)$ have opposite sign.

point c between ℓ_k and r_k in the k-th iteration is to the root x^* that we hope to compute. This analysis will provide a baseline for comparison to other methods.

More broadly, suppose we can establish an error bound E_k such that the estimate x_k of the root x^* during the k-th iteration of root-finding satisfies $|x_k - x^*| < E_k$. Any algorithm with $E_k \to 0$ is *convergent*. Assuming a root-finding algorithm is convergent, however, the primary property of interest is the *convergence rate*, characterizing the rate at which E_k shrinks.

For bisection, since during each iteration c_k and x^* are in the interval $[\ell_k, r_k]$, an upper bound of error is given by $E_k \equiv |r_k - \ell_k|$. Since we divide the interval in half each iteration, we can reduce our error bound by half in each iteration: $E_{k+1} = {}^1\!/_2 E_k$. Since E_{k+1} is linear in E_k, we say that bisection exhibits *linear* convergence.

In exchange for unconditional linear convergence, bisection requires initial estimates of ℓ and r bracketing a root. While some heuristic search methods exist for finding a bracketing interval, unless more is known about the form of f, finding this pair may be nearly as difficult as computing a root! In this case, bisection might be thought of as a method for *refining* a root estimate rather than for global search.

8.1.3 Fixed Point Iteration

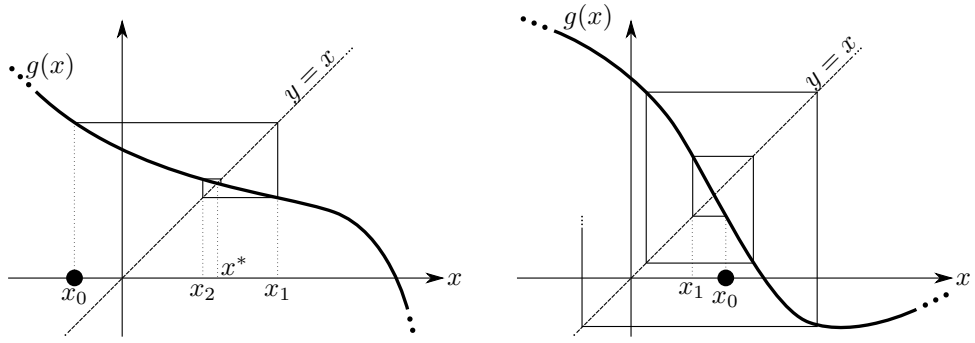
Bisection is guaranteed to converge to a root of any continuous function f, but if we know more about f we can formulate algorithms that converge more quickly.

As an example, suppose we wish to find x^* satisfying $g(x^*) = x^*$; this setup is equivalent to root-finding since solving $g(x^*) = x^*$ is the same as solving $g(x^*) - x^* = 0$. As an additional piece of information, however, we also might know that g is Lipschitz with constant $0 \leq c < 1$ (see §8.1.1). This condition defines g as a *contraction*, since $|g(x) - g(y)| < |x - y|$ for any x, y.

The system $g(x) = x$ suggests a potential solution method:

1. Take x_0 to be an initial guess of x^*.
2. Iterate $x_k = g(x_{k-1})$.

If this iteration converges, the result is a *fixed point* of g satisfying the criteria above.



(a) Convergence (b) Divergence

Figure 8.2 Convergence of fixed point iteration. Fixed point iteration searches for the intersection of $g(x)$ with the line $y = x$ by iterating $x_k = g(x_{k-1})$. One way to visualize this method on the graph of $g(x)$ visualized above is that it alternates between moving horizontally to the line $y = x$ and vertically to the position $g(x)$. Fixed point iteration (a) converges when the slope of $g(x)$ is small and (b) diverges otherwise.

When $c < 1$, the Lipschitz property ensures convergence to a root if one exists. To verify this statement, if $E_k = |x_k - x^*|$, then we have the following property:

$$\begin{aligned} E_k &= |x_k - x^*| \\ &= |g(x_{k-1}) - g(x^*)| \text{ by design of the iterative scheme and definition of } x^* \\ &\leq c|x_{k-1} - x^*| \text{ since } g \text{ is Lipschitz} \\ &= cE_{k-1}. \end{aligned}$$

Applying this statement inductively shows $E_k \leq c^k E_0 \to 0$ as $k \to \infty$.

If g is Lipschitz with constant $c < 1$ in a *neighborhood* $[x^* - \delta, x^* + \delta]$, then so long as x_0 is chosen in this interval, fixed point iteration will converge. This is true since our expression for E_k above shows that it shrinks each iteration. When the Lipschitz constant is too large—or equivalently, when g has large slope—fixed point iteration diverges. Figure 8.2 visualizes the two possibilities.

One important case occurs when g is C^1 and $|g'(x^*)| < 1$. By continuity of g' in this case, there are values $\varepsilon, \delta > 0$ such that $|g'(x)| < 1 - \varepsilon$ for any $x \in (x^* - \delta, x^* + \delta)$. (This statement is hard to parse: Make sure you understand it!) Take any $x, y \in (x^* - \delta, x^* + \delta)$. Then,

$$\begin{aligned} |g(x) - g(y)| &= |g'(\theta)| \cdot |x - y| \text{ by the Mean Value Theorem, for some } \theta \in [x, y] \\ &< (1 - \varepsilon)|x - y|. \end{aligned}$$

This argument shows that g is Lipschitz with constant $1-\varepsilon < 1$ in the interval $(x^*-\delta, x^*+\delta)$. Applying our earlier discussion, when g is continuously differentiable and $g'(x^*) < 1$, fixed point iteration will converge to x^* when the initial guess x_0 is close by.

So far, we have little reason to use fixed point iteration: We have shown it is guaranteed to converge only when g is Lipschitz, and our argument about the E_k's shows linear convergence, like bisection. There is one case, however, in which fixed point iteration provides an advantage.

Suppose g is differentiable with $g'(x^*) = 0$. Then, the first-order term vanishes in the Taylor series for g, leaving behind:

$$g(x_k) = g(x^*) + \frac{1}{2}g''(x^*)(x_k - x^*)^2 + O\left((x_k - x^*)^3\right).$$

In this case,

$$\begin{aligned}
E_k &= |x_k - x^*| \\
&= |g(x_{k-1}) - g(x^*)| \text{ as before} \\
&= \frac{1}{2}|g''(x^*)|(x_{k-1} - x^*)^2 + O((x_{k-1} - x^*)^3) \text{ from the Taylor argument} \\
&\leq \frac{1}{2}(|g''(x^*)| + \varepsilon)(x_{k-1} - x^*)^2 \text{ for some } \varepsilon \text{ so long as } x_{k-1} \text{ is close to } x^* \\
&= \frac{1}{2}(|g''(x^*)| + \varepsilon)E_{k-1}^2.
\end{aligned}$$

By this chain of inequalities, in this case E_k is *quadratic* in E_{k-1}, so we say fixed point iteration can have *quadratic convergence*. This implies that $E_k \to 0$ much faster, needing fewer iterations to reach a reasonable root approximation.

Example 8.2 (Fixed point iteration)**.** We can apply fixed point iteration to solving $x = \cos x$ by iterating $x_{k+1} = \cos x_k$. A numerical example starting from $x_0 = 0$ proceeds as follows:

k	0	1	2	3	4	5	6	7	8	9
x_k	0	1.000	0.540	0.858	0.654	0.793	0.701	0.764	0.722	0.750

In this case, fixed point iteration converges linearly to the root $x^* \approx 0.739085$.

The root-finding problem $x = \sin x^2$ satisfies the condition for quadratic convergence near $x^* = 0$. For this reason, fixed point iteration $x_{k+1} = \sin x_k^2$ starting at $x_0 = 1$ converges more quickly to the root:

k	0	1	2	3	4	5	6	7	8	9
x_k	1	0.841	0.650	0.410	0.168	0.028	0.001	0.000	0.000	0.000

Finally, the roots of $x = e^x + e^{-x} - 5$ do *not* satisfy convergence criteria for fixed point iteration. Iterates of the failed fixed point scheme $x_{k+1} = e^{x_k} + e^{-x_k} - 5$ starting at $x_0 = 1$ are shown below:

k	0	1	2	3	4	5	6	7
x_k	1	−1.914	1.927	2.012	2.609	8.660	5760.375	⋯

8.1.4 Newton's Method

We tighten our class of functions once more to derive a root-finding algorithm based more fundamentally on a differentiability assumption, this time with consistent quadratic convergence. We will attempt to solve $f(x^*) = 0$ rather than finding fixed points, with the assumption that $f \in C^1$—a slightly tighter condition than Lipschitz.

Since f is differentiable, it can be approximated near $x_k \in \mathbb{R}$ using a tangent line:

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k).$$

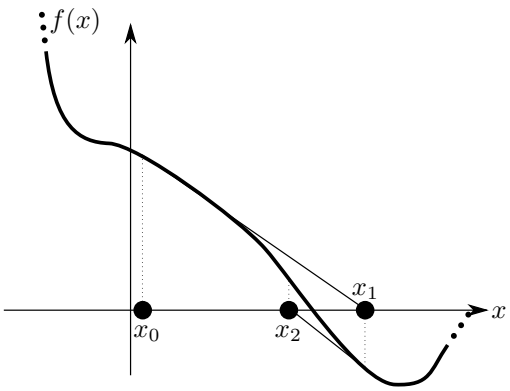


Figure 8.3 Newton's method iteratively approximates $f(x)$ with tangent lines to find roots of a differentiable function $f(x)$.

Setting the expression on the right equal to zero and solving for x provides an approximation x_{k+1} of the root:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

In reality, x_{k+1} may not satisfy $f(x_{k+1}) = 0$, but since it is the root of an approximation of f we might hope that it is closer to x^* than x_k. If this is true, then iterating this formula should give x_k's that get closer and closer to x^*. This technique is known as *Newton's method* for root-finding, and it amounts to repeatedly solving linear approximations of the original nonlinear problem. It is illustrated in Figure 8.3.

If we define

$$g(x) = x - \frac{f(x)}{f'(x)},$$

then Newton's method amounts to fixed point iteration on g. Differentiating,

$$\begin{aligned} g'(x) &= 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} \text{ by the quotient rule} \\ &= \frac{f(x)f''(x)}{f'(x)^2} \text{ after simplification.} \end{aligned}$$

Suppose x^* is a *simple* root of $f(x)$, meaning $f'(x^*) \neq 0$. Using this formula, $g'(x^*) = 0$, and by our analysis of fixed point iteration in §8.1.3, Newton's method must converge *quadratically* to x^* when starting from a sufficiently close x_0. When x^* is not simple, however, convergence of Newton's method can be linear or worse.

The derivation of Newton's method via linear approximation suggests other methods using more terms in the Taylor series. For instance, "Halley's method" also makes use of f'' via quadratic approximation, and more general "Householder methods" can include an arbitrary number of derivatives. These techniques offer higher-order convergence at the cost of having to evaluate many derivatives and the possibility of more exotic failure modes. Other methods replace Taylor series with alternative approximations; for example, "linear fractional interpolation" uses rational functions to better approximate functions with asymptotes.

Example 8.3 (Newton's method)**.** The last part of Example 8.2 can be expressed as a root-finding problem on $f(x) = e^x + e^{-x} - 5 - x$. The derivative of $f(x)$ in this case is $f'(x) = e^x - e^{-x} - 1$, so Newton's method can be written

$$x_{k+1} = x_k - \frac{e^{x_k} + e^{-x_k} - 5 - x_k}{e^{x_k} - e^{-x_k} - 1}.$$

This iteration quickly converges to a root starting from $x_0 = 2$:

k	0	1	2	3	4
x_k	2	1.9161473	1.9115868	1.9115740	1.9115740

Example 8.4 (Newton's method failure)**.** Suppose $f(x) = x^5 - 3x^4 + 25$. Newton's method applied to this function gives the iteration

$$x_{k+1} = x_k - \frac{x_k^5 - 3x_k^4 + 25}{5x_k^4 - 12x^3}.$$

These iterations converge when x_0 is sufficiently close to the root $x^* \approx -1.5325$. For instance, the iterates starting from $x_0 = -2$ are shown below:

k	0	1	2	3	4
x_k	-2	-1.687500	-1.555013	-1.533047	-1.532501

Farther away from this root, however, Newton's method can fail. For instance, starting from $x_0 = 0.25$ gives a divergent set of iterates:

k	0	1	2	3	4
x_k	0.25	149.023256	119.340569	95.594918	76.599025

8.1.5 Secant Method

One concern about Newton's method is the cost of evaluating f and its derivative f'. If f is complicated, we may wish to minimize the number of times we have to evaluate either of these functions. Higher orders of convergence for root-finding alleviate this problem by reducing the number of iterations needed to approximate x^*, but we also can design numerical methods that explicitly avoid evaluating costly derivatives.

Example 8.5 (Rocket design)**.** Suppose we are designing a rocket and wish to know how much fuel to add to the engine. For a given number of gallons x, we can write a function $f(x)$ giving the maximum height of the rocket during flight; our engineers have specified that the rocket should reach a height h, so we need to solve $f(x) = h$. Evaluating $f(x)$ involves simulating a rocket as it takes off and monitoring its fuel consumption, which is an expensive proposition. Even if f is differentiable, we might not be able to evaluate f' in a practical amount of time.

One strategy for designing lower-impact methods is to reuse data as much as possible. For instance, we could approximate the derivative f' appearing in Newton's method as follows:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

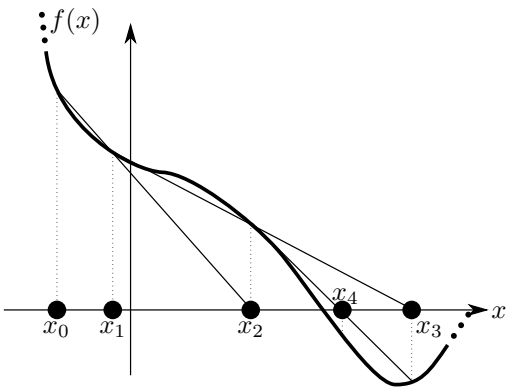


Figure 8.4 The secant method is similar to Newton's method (Figure 8.3) but approximates tangents to $f(x)$ as the lines through previous iterates. It requires both x_0 and x_1 for initialization.

Since we had to compute $f(x_{k-1})$ in the previous iteration anyway, we reuse this value to approximate the derivative for the next one. This approximation works well when x_k's are near convergence and close to one another. Plugging it into Newton's method results in a new scheme known as the *secant method*, illustrated in Figure 8.4:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}.$$

The user must provide two initial guesses x_0 and x_1 or can run a single iteration of Newton to get it started.

Analyzing the secant method is more involved than the other methods we have considered because it uses both $f(x_k)$ and $f(x_{k-1})$; proof of its convergence is outside the scope of our discussion. Error analysis reveals that the secant method decreases error at a rate of $(1+\sqrt{5})/2$ (the "Golden Ratio"), which is between linear and quadratic. Since convergence is *close* to that of Newton's method without the need for evaluating f', the secant method is a strong alternative.

Example 8.6 (Secant method)**.** Suppose $f(x) = x^4 - 2x^2 - 4$. Iterates of Newton's method for this function are given by

$$x_{k+1} = x_k - \frac{x_k^4 - 2x_k^2 - 4}{4x_k^3 - 4x_k}.$$

Contrastingly, iterates of the secant method for the same function are given by

$$x_{k+1} = x_k - \frac{(x_k^4 - 2x_k^2 - 4)(x_k - x_{k-1})}{(x_k^4 - 2x_k^2 - 4) - (x_{k-1}^4 - 2x_{k-1}^2 - 4)}.$$

By construction, a less expensive way to compute these iterates is to save and reuse $f(x_{k-1})$ from the previous iteration. We can compare the two methods starting from $x_0 = 3$; for the secant method we also choose $x_{-1} = 2$:

k	0	1	2	3	4	5	6
x_k (Newton)	3	2.385417	2.005592	1.835058	1.800257	1.798909	1.798907
x_k (secant)	3	1.927273	1.882421	1.809063	1.799771	1.798917	1.798907

The two methods exhibit similar convergence on this example.

8.1.6 Hybrid Techniques

With additional engineering, we can combine the advantages of different root-finding algorithms. For instance, we might make the following observations:

- *Bisection* is guaranteed to converge, but only at a linear rate.
- The *secant method* has a faster rate of convergence, but it may not converge at all if the initial guess x_0 is far from the root x^*.

Suppose we have bracketed a root of $f(x)$ in the interval $[\ell_k, r_k]$. Given the iterates x_k and x_{k-1}, we could take the next estimate x_{k+1} to be either of the following:

- The next secant method iterate, if it is contained in (ℓ_k, r_k).
- The midpoint $\ell_k + r_k/2$ otherwise.

This combination of the secant method and bisection guarantees that $x_{k+1} \in (\ell_k, r_k)$. Regardless of the choice above, we can update the bracket containing the root to $[\ell_{k+1}, r_{k+1}]$ by examining the sign of $f(x_{k+1})$.

The algorithm above, called "Dekker's method," attempts to combine the unconditional convergence of bisection with the stronger root estimates of the secant method. In many cases it is successful, but its convergence rate is somewhat difficult to analyze. Specialized failure modes can reduce this method to linear convergence or worse: In some cases, bisection can converge *more* quickly! Other techniques, e.g., "Brent's method," make bisection steps more often to strengthen convergence and can exhibit guaranteed behavior at the cost of a more complex implementation.

8.1.7 Single-Variable Case: Summary

We only have scratched the surface of the one-dimensional root-finding problem. Many other iterative schemes for root-finding exist, with different guarantees, convergence rates, and caveats. Starting from the methods above, we can make a number of broader observations:

- To support arbitrary functions f that may not have closed-form solutions to $f(x^*) = 0$, we use iterative algorithms generating approximations that get closer and closer to the desired root.
- We wish for the sequence x_k of root estimates to reach x^* as quickly as possible. If E_k is an error bound with $E_k \to 0$ as $k \to \infty$, then we can characterize the order of convergence using classifications like the following:
  1. Linear convergence: $E_{k+1} \leq CE_k$ for some $C < 1$.
  2. Superlinear convergence: $E_{k+1} \leq CE_k^r$ for $r > 1$; we do not require $C < 1$ since if E_k is small enough, the r-th power of E_k can cancel the effects of C.
  3. Quadratic convergence: $E_{k+1} \leq CE_k^2$.
  4. Cubic convergence: $E_{k+1} \leq CE_k^3$ (and so on).
- A method might converge quickly, needing fewer iterations to get sufficiently close to x^*, but each individual iteration may require additional computation time. In this case, it may be preferable to do more iterations of a simpler method than fewer iterations of a more complex one. This idea is further explored in Exercise 8.1.

8.2 MULTIVARIABLE PROBLEMS

Some applications may require solving the multivariable problem $f(\vec{x}) = \vec{0}$ given a function $f : \mathbb{R}^n \to \mathbb{R}^m$. We have already seen one instance of this problem when solving $A\vec{x} = \vec{b}$, which is equivalent to finding roots of $f(\vec{x}) \equiv A\vec{x} - \vec{b}$, but the general case is considerably more difficult. Strategies like bisection are challenging to extend since we now must guarantee that m different functions all equal zero *simultaneously*.

8.2.1 Newton's Method

One of our single-variable strategies extends in a straightforward way. Recall from §1.4.2 that for a differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$ we can define the *Jacobian* matrix giving the derivative of each component of f in each of the coordinate directions:

$$(Df)_{ij} \equiv \frac{\partial f_i}{\partial x_j}.$$

We can use the Jacobian of f to extend our derivation of Newton's method to multiple dimensions. In more than one dimension, a first-order approximation of f is given by

$$f(\vec{x}) \approx f(\vec{x}_k) + Df(\vec{x}_k) \cdot (\vec{x} - \vec{x}_k).$$

Substituting the desired condition $f(\vec{x}) = \vec{0}$ yields the following linear system determining the next iterate $\vec{x}_{k+1}$:

$$Df(\vec{x}_k) \cdot (\vec{x}_{k+1} - \vec{x}_k) = -f(\vec{x}_k).$$

When Df is square and invertible, requiring $n = m$, we obtain the iterative formula for a multidimensional version of Newton's method:

$$\vec{x}_{k+1} = \vec{x}_k - [Df(\vec{x}_k)]^{-1} f(\vec{x}_k),$$

where as always we do not explicitly compute the matrix $[Df(\vec{x}_k)]^{-1}$ but rather solve a linear system, e.g., using the techniques from Chapter 3. When $m < n$, this equation can be solved using the pseudoinverse to find one of potentially many roots of f; when $m > n$, one can attempt least-squares, but the existence of a root and convergence of this technique are both unlikely.

An analogous multidimensional argument to that in §8.1.3 shows that fixed-point methods like Newton's method iterating $\vec{x}_{k+1} = g(\vec{x}_k)$ converge when the largest-magnitude eigenvalue of Dg has absolute value less than 1 (Exercise 8.2). A derivation identical to the one-dimensional case in §8.1.4 then shows that Newton's method in multiple variables can have quadratic convergence near roots $\vec{x}^*$ for which $Df(\vec{x}^*)$ is nonsingular.

8.2.2 Making Newton Faster: Quasi-Newton and Broyden

As m and n increase, Newton's method becomes very expensive. For each iteration, a *different* matrix $Df(\vec{x}_k)$ must be inverted. Since it changes in each iteration, factoring $Df(\vec{x}_k) = L_k U_k$ does not help.

Quasi-Newton algorithms apply various approximations to reduce the cost of individual iterations. One approach extends the secant method beyond one dimension. Just as the secant method contains the same division operation as Newton's method, such secant-like approximations will not necessarily alleviate the need to invert a matrix. Instead, they make it possible to carry out root-finding without explicitly calculating the Jacobian Df.

An extension of the secant method to multiple dimensions will require careful adjustment, however, since divided differences yield a single value rather than a full approximate Jacobian matrix.

The directional derivative of f in the direction $\vec{v}$ is given by $D_{\vec{v}}f = Df \cdot \vec{v}$. To imitate the secant method, we can use this scalar value to our advantage by requiring that the Jacobian approximation J satisfies

$$J_k \cdot (\vec{x}_k - \vec{x}_{k-1}) = f(\vec{x}_k) - f(\vec{x}_{k-1}).$$

This formula does not determine the action of J on any vector perpendicular to $\vec{x}_k - \vec{x}_{k-1}$, so we need additional approximation assumptions to describe a complete root-finding algorithm.

One algorithm using the approximation above is *Broyden's method*, which maintains not only an estimate $\vec{x}_k$ of $\vec{x}^*$ but also a full matrix J_k estimating a Jacobian at $\vec{x}_k$ that satisfies the condition above. Initial estimates J_0 and $\vec{x}_0$ both must be supplied by the user; commonly, we approximate $J_0 = I_{n\times n}$ in the absence of more information.

Suppose we have an estimate J_{k-1} of the Jacobian at $\vec{x}_{k-1}$ left over from the previous iteration. We now have a new data point $\vec{x}_k$ at which we have evaluated $f(\vec{x}_k)$, so we would like to update J_{k-1} to a new matrix J_k taking into account this new piece of information. Broyden's method applies the directional derivative approximation above to finding J_k while keeping it as similar as possible to J_{k-1} by solving the following optimization problem:

$$\begin{array}{ll} \text{minimize}_{J_k} & \|J_k - J_{k-1}\|_{\text{Fro}}^2 \\ \text{subject to} & J_k \cdot (\vec{x}_k - \vec{x}_{k-1}) = f(\vec{x}_k) - f(\vec{x}_{k-1}). \end{array}$$

To solve this problem, define $\Delta J \equiv J_k - J_{k-1}$, $\Delta\vec{x} \equiv \vec{x}_k - \vec{x}_{k-1}$, and $\vec{d} \equiv f(\vec{x}_k) - f(\vec{x}_{k-1}) - J_{k-1} \cdot \Delta\vec{x}$. Making these substitutions provides an alternative optimization problem:

$$\begin{array}{ll} \text{minimize}_{\Delta J} & \|\Delta J\|_{\text{Fro}}^2 \\ \text{subject to} & \Delta J \cdot \Delta\vec{x} = \vec{d}. \end{array}$$

If we take $\vec{\lambda}$ to be a Lagrange multiplier, this minimization is equivalent to finding critical points of the Lagrangian Λ:

$$\Lambda = \|\Delta J\|_{\text{Fro}}^2 + \vec{\lambda}^\top(\Delta J \cdot \Delta\vec{x} - \vec{d}).$$

Differentiating with respect to an unknown element $(\Delta J)_{ij}$ shows:

$$0 = \frac{\partial\Lambda}{\partial(\Delta J)_{ij}} = 2(\Delta J)_{ij} + \lambda_i(\Delta\vec{x})_j \implies \Delta J = -\frac{1}{2}\vec{\lambda}(\Delta\vec{x})^\top.$$

Substituting into $\Delta J \cdot \Delta\vec{x} = \vec{d}$ shows $\vec{\lambda}(\Delta\vec{x})^\top(\Delta\vec{x}) = -2\vec{d}$, or equivalently $\vec{\lambda} = -2\vec{d}/\|\Delta\vec{x}\|_2^2$. Finally, we substitute into the Lagrange multiplier expression to find:

$$\Delta J = -\frac{1}{2}\vec{\lambda}(\Delta\vec{x})^\top = \frac{\vec{d}(\Delta\vec{x})^\top}{\|\Delta\vec{x}\|_2^2}.$$

Expanding back to the original notation shows:

$$\begin{aligned} J_k &= J_{k-1} + \Delta J \\ &= J_{k-1} + \frac{\vec{d}(\Delta\vec{x})^\top}{\|\Delta\vec{x}\|_2^2} \\ &= J_{k-1} + \frac{(f(\vec{x}_k) - f(\vec{x}_{k-1}) - J_{k-1}\cdot\Delta\vec{x})}{\|\vec{x}_k - \vec{x}_{k-1}\|_2^2}(\vec{x}_k - \vec{x}_{k-1})^\top. \end{aligned}$$

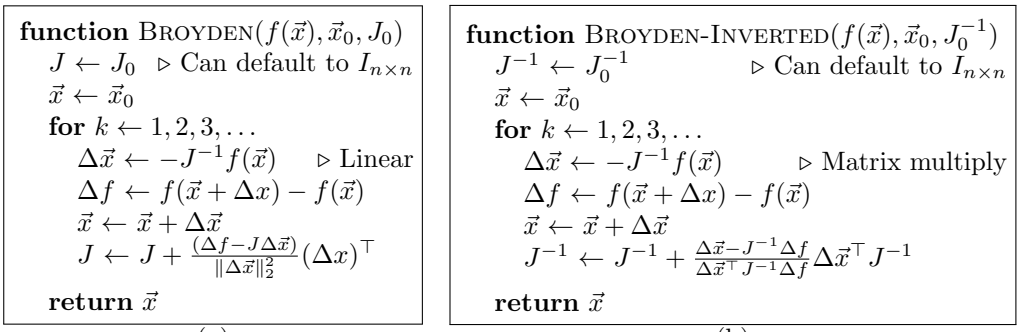


Figure 8.5 (a) Broyden's method as described in §8.2.2 requires solving a linear system of equations, but the formula from Exercise 8.7 yields (b) an equivalent method using only matrix multiplication by updating the inverse matrix J^{-1} directly instead of J.

Broyden's method alternates between this update and the corresponding Newton step $\vec{x}_{k+1} = \vec{x}_k - J_k^{-1}f(\vec{x}_k)$.

Additional efficiency in some cases can be gained by keeping track of the matrix J_k^{-1} explicitly rather than the matrix J_k, which can be updated using a similar formula and avoids the need to solve any linear systems of equations. This possibility is explored via the Sherman-Morrison update formula in Exercise 8.7. Both versions of the algorithm are shown in Figure 8.5.

8.3 CONDITIONING

We already showed in Example 2.10 that the condition number of root-finding in a single variable is:

$$\text{cond}_{x^*} f = \frac{1}{|f'(x^*)|}.$$

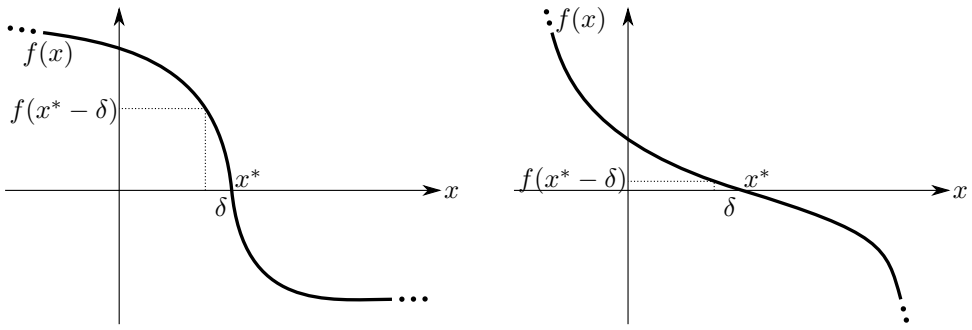
As shown in Figure 8.6, this condition number shows that the best possible situation for root-finding occurs when f is changing rapidly near x^*, since in this case perturbing x^* will make f take values far from 0.

Applying an identical argument when f is multidimensional gives a condition number of $\|Df(\vec{x}^*)\|^{-1}$. When Df is not invertible, the condition number is *infinite*. This degeneracy occurs because perturbing $\vec{x}^*$ preserves $f(\vec{x}) = \vec{0}$ to first order, and indeed such a condition can create challenging root-finding cases similar to that shown in Figure 8.6(b).

8.4 EXERCISES

8.1 Suppose it takes processor time t to evaluate $f(x)$ or $f'(x)$ given $x \in \mathbb{R}$. So, computing the pair $(f(x), f'(x))$ takes time $2t$. For this problem, assume that individual arithmetic operations take negligible amounts of processor time compared to t.

(a) Approximately how much time does it take to carry out k iterations of Newton's method on $f(x)$? Approximately how much time does it take to carry out k iterations of the secant method on $f(x)$?



(a) Good conditioning (b) Poor conditioning

Figure 8.6 Intuition for the conditioning of finding roots of a function $f(x)$. (a) When the slope at the root x^* is large, the problem is well-conditioned because moving a small distance δ away from x^* makes the value of f change by a large amount. (b) When the slope at x^* is smaller, values of $f(x)$ remain close to zero as we move away from the root, making it harder to pinpoint the exact location of x^*.

(b) Why might the secant method be preferable in this case?

[DH] 8.2 Recall from §8.1.3 the proof of conditions under which single-variable fixed point iteration converges. Consider now the multivariable fixed point iteration scheme $\vec{x}_{k+1} \equiv g(\vec{x}_k)$ for $g : \mathbb{R}^n \to \mathbb{R}^n$.

(a) Suppose that $g \in C^1$ and that $\vec{x}_k$ is within a small neighborhood of a fixed point $\vec{x}^*$ of g. Suggest a condition on the Jacobian Dg of g that guarantees g is Lipschitz in this neighborhood.

(b) Using the previous result, derive a bound for the error of $\vec{x}_{k+1}$ in terms of the error of $\vec{x}_k$ and the Jacobian of g.

(c) Show a condition on the eigenvalues of Dg that guarantees convergence of multivariable fixed point iteration.

(d) How does the rate of convergence change if $Dg(\vec{x}^*) = 0$?

[DH] 8.3 Which method would you recommend for finding the root of $f : \mathbb{R} \to \mathbb{R}$ if all you know about f is that:

(a) $f \in C^1$ and f' is inexpensive to evaluate;

(b) f is Lipschitz with constant c satisfying $0 \leq c \leq 1$;

(c) $f \in C^1$ and f' is costly to evaluate; or

(d) $f \in C^0 \backslash C^1$, the continuous but non-differentiable functions.

[DH] 8.4 Provide an example of root-finding problems that satisfy the following criteria:

(a) Can be solved by bisection but not by fixed point iteration

(b) Can be solved using fixed point iteration, but not using Newton's method

8.5 Is Newton's method guaranteed to have quadratic convergence? Why?

[DH] 8.6 Suppose we wish to compute $\sqrt[r]{y}$ for a given $y > 0$. Using the techniques from this chapter, derive a quadratically convergent iterative method that finds this root given a sufficiently close initial guess.

8.7 In this problem, we show how to carry out Broyden's method for finding roots without solving linear systems of equations.

(a) Verify the Sherman-Morrison formula, for invertible $A \in \mathbb{R}^{n\times n}$ and vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$:

$$(A + \vec{u}\vec{v}^\top)^{-1} = A^{-1} - \frac{A^{-1}\vec{u}\vec{v}^\top A^{-1}}{1 + \vec{v}^\top A^{-1}\vec{u}}.$$

(b) Use this formula to show that the algorithm in Figure 8.5(b) is equivalent to Broyden's method as described in §8.2.2.

8.8 In this problem, we will derive a technique known as Newton-Raphson division. Thanks to its fast convergence, it is often implemented in hardware for IEEE-754 floating-point arithmetic.

(a) Show how the reciprocal $\frac{1}{a}$ of $a \in \mathbb{R}$ can be computed iteratively using Newton's method. Write your iterative formula in a way that requires at most two multiplications, one addition or subtraction, and no divisions.

(b) Take x_k to be the estimate of $\frac{1}{a}$ during the k-th iteration of Newton's method. If we define $\varepsilon_k \equiv ax_k - 1$, show that $\varepsilon_{k+1} = -\varepsilon_k^2$.

(c) Approximately how many iterations of Newton's method are needed to compute $\frac{1}{a}$ within d binary decimal points? Write your answer in terms of ε_0 and d, and assume $|\varepsilon_0| < 1$.

(d) Is this method always convergent regardless of the initial guess of $\frac{1}{a}$?

8.9 (LSQI, [50]) In this problem, we will develop a method for solving least-squares with a quadratic inequality constraint:

$$\min_{\|\vec{x}\|_2 \le 1} \|A\vec{x} - \vec{b}\|_2.$$

You can assume the least-squares system $A\vec{x} \approx \vec{b}$, where $A \in \mathbb{R}^{m\times n}$ with $m > n$, is overdetermined.

(a) The optimal $\vec{x}$ either satisfies $\|\vec{x}\|_2 < 1$ or $\|\vec{x}\|_2 = 1$. Explain how to distinguish between the two cases, and give a formula for $\vec{x}$ when $\|\vec{x}\|_2 < 1$.

(b) Suppose we are in the $\|\vec{x}\|_2 = 1$ case. Show that there exists $\lambda \in \mathbb{R}$ such that $(A^\top A + \lambda I_{n\times n})\vec{x} = A^\top \vec{b}$.

(c) Define $f(\lambda) \equiv \|\vec{x}(\lambda)\|_2^2 - 1$, where $\vec{x}(\lambda)$ is the solution to the system $(A^\top A + \lambda I_{n\times n})\vec{x} = A^\top \vec{b}$ for fixed $\lambda \ge 0$. Assuming that the optimal $\vec{x}$ for the original optimization problem satisfies $\|\vec{x}\|_2 = 1$, show $f(0) \ge 0$ and that $f(\lambda) < 0$ for sufficiently large $\lambda > 0$.

(d) Propose a strategy for the $\|\vec{x}\|_2 = 1$ case using root-finding.

8.10 (Proposed by A. Nguyen.) Suppose we have a polynomial $p(x) = a_k x^k + \cdots + a_1 x + a_0$. You can assume $a_k \neq 0$ and $k \geq 1$.

(a) Suppose the derivative $p'(x)$ has no roots in (a, b). How many roots can $p(x)$ have in this interval?

(b) Using the result of Exercise 8.10a, propose a recursive algorithm for estimating all the real roots of $p(x)$. Assume we know that the roots of $p(x)$ are at least ε apart and that they are contained with an interval $[a, b]$.

8.11 Root-finding for complex- or real-valued polynomials is closely linked to the eigenvalue problem considered in Chapter 6.

(a) Give a matrix A whose eigenvalues are the roots of a given polynomial $p(x) = a_k x^k + \cdots + a_1 x + a_0$; you can assume $p(x)$ has no repeated roots.

(b) Show that the eigenvalues of a matrix $A \in \mathbb{R}^{n \times n}$ are the roots of a polynomial function. Is it advisable to use root-finding algorithms from this chapter for the eigenvalue problem?