Chapter 19
Nonlinear Systems

Nonlinearity is ubiquitous in physical phenomena. Fluid mechanics, elasticity, relativity, chemical reactions, combustion, ecology, biomechanics, and many, many others are all governed by inherently nonlinear equations. (The one notable exception is quantum mechanics, which is a fundamentally linear theory. More recent attempts at grand unification of all fundamental physical theories, such as string theory and conformal field theory, do venture into the nonlinear realm.) For this reason, an increasingly large fraction of modern mathematical research is devoted to the analysis of nonlinear systems. The advent of powerful computers has finally placed nonlinearity within our grasp, and thereby fomented a revolution in our understanding and development of nonlinear mathematics. Indeed, many of the most important modern analytical techniques drew their inspiration from early computer forays into the uncharted nonlinear wilderness.

Why, then, have we spent the overwhelming majority of this text developing purely linear mathematics? The facile answer, of course, is that nonlinear systems are vastly more difficult to analyze. In the nonlinear regime, many basic questions remain unanswered; existence and uniqueness of solutions are not guaranteed; explicit formulae are difficult to come by; linear superposition is no longer available; numerical approximations are not always sufficiently accurate; etc., etc. But, a more intelligent answer is that, without a proper understanding of linear phenomena and linear mathematics, one has no foundation upon which to erect a nonlinear analysis. Therefore, in an introductory text on applied mathematics, we are forced to develop in detail the proper linear foundations to aid us when we confront the nonlinear beast.

Moreover, many important physical systems are “weakly nonlinear”, in the sense that, while nonlinear effects do play an essential role, the linear terms dominate the system, and so, to a first approximation, the system is close to linear. As a result, the underlying nonlinear phenomena can be understood by suitably perturbing their linear approximations. Historically, while certain nonlinear problems date back to Newton (for example the $n$ body problem arising in celestial mechanics and planetary motion), significant progress in understanding weak nonlinearities only began after computers became sufficiently powerful tools. The truly nonlinear regime is, even today, only sporadically modeled and even less well understood. Despite dramatic advances in both hardware and mathematical algorithms, many nonlinear systems, for instance Einsteinian gravitation, still remain beyond the capabilities of today’s computers and algorithms.

Space limitations imply that we can only provide a brief overview of some of the key ideas and phenomena that arise when venturing into the nonlinear realm. This chapter is devoted to the study of nonlinear functions and equations. In the remaining chapters, we
shall ascend the nonlinear “dimensional ladder”, passing from equilibrium to dynamics and from discrete to continuous, mimicking our linear ascent that guided the logical progression in the preceding chapters of the text.

We begin with an analysis of the iteration of nonlinear functions. Building on our experience with iteration of linear systems, we will discover that functional iteration, when it converges, provides a powerful mechanism for solving equations and optimization. When it fails to converge, even very simple nonlinear iterations can lead to remarkably complex, chaotic behavior. The second section is devoted to basic solution techniques for nonlinear systems, and includes the bisection method, iterative methods, and the powerful Newton method. The third section is devoted to optimization, i.e., the minimization of nonlinear functions on finite-dimensional spaces. As we know, the equilibrium configurations of discrete mechanical systems are minimizers of the potential energy in the system. The locations where the gradient of the function vanishes are the critical points, and include the local minima and maxima as well as non-optimizing saddle points. Nondegenerate critical points are classified by a second derivative test based on Hessian matrix. These results from multivariable calculus will be developed in a form that readily generalizes to minimization problems on infinite-dimensional function space, to be presented in Chapter 21. Numerical optimization procedures rely on iterative procedures, and we present those connected with a gradient descent approach.

19.1. Iteration of Functions.

Iteration, or repeated application of a function, plays an essential role in the modern theories of dynamical systems. Iteration can be regarded as a discrete dynamical system, in which the continuous time variable has been “quantized”. Even iterating a very simple quadratic function leads to an amazing variety of phenomena, including convergence, period doubling, and chaos. Discrete dynamical systems arise not just in mathematics, but also underlie the theory of growth and decay of biological populations, predator-prey models, spread of communicable diseases such as AIDS, and host of other natural phenomena. Moreover, many numerical solution methods — for systems of algebraic equations, ordinary differential equations, partial differential equations and so on — rely in essence on an iterative method, and so the basic results on function iteration play a key role in the analysis of convergence and efficiency of such numerical techniques.

In general, an iterative system of the form

\[ u^{(k+1)} = g(u^{(k)}) \]  

(19.1)
is also known as a discrete dynamical system. A solution is a discrete collection of points \( u^{(k)} \) in which the index \( k = 0, 1, 2, 3, \ldots \) takes on non-negative integer values. One might also consider negative integral values \( k = -1, -2, \ldots \) of the index, but we will not. The superscripts on \( u^{(k)} \) refer to the iteration number, and do not denote derivatives. The index \( k \) may be viewed as the discrete “time” for the system, indicating the number of days, years, seconds, etc.

The function\(^\dagger\) \( g : \mathbb{R}^n \to \mathbb{R}^n \) is usually assumed to be continuous. Later on we shall also

\(^\dagger\) Complex iteration is based on a complex-valued function \( g : \mathbb{C}^n \to \mathbb{C}^n \).
require that $g$ be reasonably smooth, meaning that it has at least one or two continuous partial derivatives everywhere. Chapter 10 dealt with the case when $g(u) = Au$ is a linear function, necessarily given by multiplication by an $n \times n$ matrix $A$. In this chapter, we allow nonlinear functions into the picture.

Once we specify an initial condition, say

$$u^{(0)} = c,$$

for the initial iterate, then the solution is easy to compute mechanically:

$$u^{(1)} = g(u^{(0)}) = g(c), \quad u^{(2)} = g(u^{(1)}) = g(g(c)), \quad u^{(3)} = g(u^{(2)}) = g(g(g(c))), \ldots$$

and so on. Therefore, unlike continuous dynamical systems, existence and uniqueness of solutions is immediate. As long as each successive iterate $u^{(k)}$ lies in the domain of definition of $g$ one merely repeats the process to produce the solution,

$$u^{(k)} = g \circ g \circ \cdots \circ g(c), \quad k = 0, 1, 2, \ldots,$$

which is obtained by composing the function $g$ with itself a total of $k$ times. In other words, the solution to a discrete dynamical system corresponds to repeatedly pushing the $g$ key on your calculator. For example, repeatedly hitting the $\sin$ key corresponds to a solution to the system $u^{(k+1)} = \sin u^{(k)}$. For simplicity, we shall tacitly assume that the function $g$ is defined on all of $\mathbb{R}^n$. Otherwise, we must always be careful that the successive iterates $u^{(k)}$ never leave the domain of definition of $g$, which would cause the iteration to break down.

While the solution to a discrete dynamical system is essentially trivial, understanding its behavior is definitely not. Sometimes the solution converges to a particular value — the key requirement for numerical solution methods. Sometimes it goes off to $\infty$, or, more precisely, $\|u^{(k)}\| \to \infty$. Sometimes the solution repeats itself after a while. And sometimes it behaves in a random, chaotic manner — all depending on the function $g$ and, at times, the initial condition $c$. Although any of these cases may appear and play a role in applications, we shall mostly concentrate upon understanding the case of convergence of the iterates.

**Definition 19.1.** A fixed point or equilibrium solution for a discrete dynamical system (19.1) is a vector $u^* \in \mathbb{R}^n$ such that

$$g(u^*) = u^*.$$  

We easily see that every fixed point provides a constant solution, namely $u^{(k)} \equiv u^*$, to the discrete dynamical system. Moreover, solutions that converge always converge to a fixed point.

**Proposition 19.2.** If a solution to a discrete dynamical system converges,

$$\lim_{k \to \infty} u^{(k)} = u^*,$$

then the limit $u^*$ is a fixed point of the system.
Proof: This is a simple consequence of the continuity of $g$. We have

$$ u^* = \lim_{k \to \infty} u^{(k+1)} = \lim_{k \to \infty} g(u^{(k)}) = g \left( \lim_{k \to \infty} u^{(k)} \right) = g(u^*), $$

the last two equalities following from the continuity of $g$. Q.E.D.

Of course, not every solution to a discrete dynamical system will necessarily converge, but Proposition 19.2 says that if it does, then it must converge to a fixed point. Thus, the goal is to understand when a solution converges, and, if so, to which fixed point — if there is more than one. (In the linear case, only the actual convergence is a significant issues since most linear systems admit exactly one fixed point, namely $u^* = 0$.) Fixed points are roughly divided into three classes: *asymptotically stable*, with the property that all nearby solutions converge to it, *stable*, with the property that all nearby solutions stay nearby, and *unstable*, almost all of whose nearby solutions diverge away from the fixed point. Thus, from a practical standpoint, convergence of the iterates of a discrete dynamical system requires asymptotic stability of the fixed point.

**Scalar Functions**

As always, the first step is to thoroughly understand the scalar case, and so we begin with a discrete dynamical system

$$ u^{(k+1)} = g(u^{(k)}), \quad u^{(0)} = c, \quad (19.5) $$
in which $g: \mathbb{R} \to \mathbb{R}$ is a continuous, scalar-valued function. As noted above, we will assume, for simplicity, that $g$ is defined everywhere, and so the iterates $u^{(0)}, u^{(1)}, u^{(2)}, \ldots$ are all well-defined.

The linear case $g(u) = a u$ was treated in Section 10.1, following (10.2). The simplest “nonlinear” case is that of an affine function

$$ g(u) = a u + b, \quad (19.6) $$
leading to an affine discrete dynamical system

$$ u^{(k+1)} = a u^{(k)} + b. \quad (19.7) $$
The only fixed point is the solution to

$$ u^* = g(u^*) = a u^* + b, \quad \text{namely,} \quad u^* = \frac{b}{1-a}. \quad (19.8) $$
The formula for $u^*$ requires that $a \neq 1$, and, indeed, the case $a = 1$ has no fixed point, as the reader can easily confirm; see Exercise 1. Since we already know the value of $u^*$, we can easily analyze the difference

$$ e^{(k)} = u^{(k)} - u^*, \quad (19.9) $$
between the iterate $u^{(k)}$ and the fixed point. The smaller $e^{(k)}$ is, the closer $u^{(k)}$ is to the desired fixed point. In many applications, the iterate $u^{(k)}$ is viewed as an approximation
to the fixed point $u^*$, and so $e^{(k)}$ is interpreted as the error in the $k^{th}$ iterate. Subtracting the fixed point equation (19.8) from the iteration equation (19.7), we find
\[ u^{(k+1)} - u^* = a(u^{(k)} - u^*). \]
Therefore the errors $e^{(k)}$ satisfy a linear iteration
\[ e^{(k+1)} = a e^{(k)}, \quad \text{and hence} \quad e^{(k)} = a^k e^{(0)}. \quad (19.10) \]
Therefore, as we already demonstrated in Section 10.1, the solutions to this scalar linear iteration converge,
\[ e^{(k)} \to 0 \quad \text{and hence} \quad u^{(k)} \to u^*, \quad \text{if and only if} \quad |a| < 1. \]
This is the criterion for asymptotic stability of the fixed point, or, equivalently, convergence of the affine iterative system (19.7). The magnitude of $|a| < 1$ determines the rate of convergence, and the closer it is to 0, the faster the iterates approach to the fixed point.

**Example 19.3.** Suppose $g(u) = \frac{1}{4} u + 2$, and so we consider the iterative scheme
\[ u^{(k+1)} = \frac{1}{4} u^{(k)} + 2. \]
Starting with the initial condition $u^{(0)} = 0$, the ensuing values are
\[
\begin{array}{cccccccc}
  k & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  u^{(k)} & 2.0 & 2.5 & 2.625 & 2.6562 & 2.6641 & 2.6660 & 2.6665 & 2.6666 \\
\end{array}
\]
Thus, after 8 iterations, the iterates have converged to the fixed point $u^* = \frac{8}{3}$ to 4 decimal places. The rate of convergence is $\frac{1}{4}$, and indeed
\[ |u^{(k)} - u^*| = \left(\frac{1}{4}\right)^k (u^{(0)} - u^*) = \frac{8}{3} \left(\frac{1}{4}\right)^k \to 0 \quad \text{as} \quad k \to \infty. \]

Let us now turn to the fully nonlinear case. In general, near a given point, any (smooth) nonlinear function can be approximated by its tangent line, which is an affine function; see Figure tl1. Therefore, if we are close to a fixed point $u^*$, then we might expect the behavior of the nonlinear system will behave very much like iteration of its affine approximation. And, indeed, this intuition turns out to be essentially correct. This result forms our first concrete example of linearization, in which the analysis of a nonlinear system is based on its linear (or, more correctly, affine) approximation.

The explicit formula for the tangent line to $g(u)$ near the fixed point $u = u^*$ is
\[ g(u) \approx g(u^*) + g'(u^*) (u - u^*) = a u + b, \quad (19.11) \]
where
\[ a = g'(u^*), \quad b = g(u^*) - g'(u^*) u^* = (1 - g'(u^*)) u^*. \]
Note that $u^* = b/(1 - a)$ remains a fixed point for the affine approximation. According to the preceding discussion, the convergence of the iterates for the affine approximation is governed by the size of the coefficient $a = g'(u^*)$. This observation inspires the key stability criterion for fixed points of scalar iterative systems.
Theorem 19.4. Suppose \( g(u) \) is a continuously differentiable scalar function. Suppose \( u^* = g(u^*) \) is a fixed point. If \( |g'(u^*)| < 1 \), then \( u^* \) is a stable fixed point, and hence any sequence of iterates \( u^{(k)} \) which starts out sufficiently close to \( u^* \) will converge to \( u^* \). On the other hand, if \( |g'(u^*)| > 1 \), then \( u^* \) is an unstable fixed point, and the only iterates which converge to it are those that land exactly on it, i.e., \( u^{(k)} = u^* \) for some \( k \geq 0 \).

Proof: The goal is to prove that the errors \( e^{(k)} = u^{(k)} - u^* \) between the \( k \)th iterate and the true fixed point tend to 0 as \( k \to \infty \). To this end, we try to estimate \( e^{(k+1)} \) in terms of \( e^{(k)} \). According to (19.5) and the Mean Value Theorem C.3 from calculus,

\[
e^{(k+1)} = u^{(k+1)} - u^* = g(u^{(k)}) - g(u^*) = g'(v)(u^{(k)} - u^*) = g'(v)e^{(k)},
\]

for some \( v \) lying between \( u^{(k)} \) and \( u^* \). By continuity, if \( |g'(u^*)| < 1 \) at the fixed point, then we can choose \( 0 < \rho < 1 \) such that

\[
|g'(v)| \leq \rho < 1 \quad \text{whenever} \quad |v - u^*| < \delta
\]

holds in a (perhaps small) interval surrounding the fixed point. If \( |e^{(k)}| = |u^{(k)} - u^*| < \delta \), then the point \( v \) in (19.12) satisfies (19.13). Therefore,

\[
|u^{(k+1)} - u^*| \leq \rho |u^{(k)} - u^*|,
\]

and hence

\[
|e^{(k+1)}| \leq \rho |e^{(k)}|.
\]

In particular, since \( \rho < 1 \), if \( |u^{(k)} - u^*| < \delta \), then \( |u^{(k+1)} - u^*| < \delta \), and hence the subsequent iterate \( u^{(k+1)} \) also lies in the interval where (19.13) holds. Iterating, we conclude that the errors satisfy

\[
e^{(k)} \leq \rho^k e^{(0)},
\]

and hence

\[
e^{(k)} = |u^{(k)} - u^*| \to 0 \quad \text{as} \quad k \to \infty,
\]

which completes the proof of the theorem in the stable case. The proof in unstable case is left as Exercise 10 for the reader. Q.E.D.

Remark: The borderline cases \( g'(u^*) = \pm 1 \) are not covered by the theorem. For a linear system, these cases are stable, but not asymptotically stable. For nonlinear systems, such borderline situations require more detailed knowledge of the nonlinear terms in order to resolve the status — stable or unstable — of the fixed point. Despite their importance in certain applications, we will not try to analyze such borderline cases any further here. From now on, we will only deal with asymptotically stable fixed points, and, for brevity, usually omit the adjective “asymptotically”.

Example 19.5. Given constants \( \epsilon, m \), the trigonometric equation

\[
u = m + \epsilon \sin u
\]

is known as Kepler’s equation. It arises in the study of planetary motion, with \( |\epsilon| < 1 \) representing the eccentricity of an elliptical planetary orbit and \( m \) its mean anomaly; see Figure Kepler 76. The desired solution \( u \) is the eccentric anomaly, and governs the motion of the planet around the ellipse. Details can be found in [76; p. 119].

The solutions to Kepler’s equation are the fixed points of the discrete dynamical system based on the function \( g(u) = m + \epsilon \sin u \). Note that

\[
|g'(u)| = |\epsilon \cos u| = |\epsilon| < 1,
\]

\(\frac{72x482}{72x482}^{0x0}\)
which automatically implies that the as yet unknown fixed point is stable. Indeed, Exercise implies that condition (19.17) is enough to prove the existence of a unique stable fixed point. In the particular case \( m = \epsilon = \frac{1}{2} \), the result of iterating \( u^{(k+1)} = \frac{1}{2} + \frac{1}{2} \sin u^{(k)} \) starting with \( u^{(0)} = 0 \) is

<table>
<thead>
<tr>
<th>( k )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u^{(k)} )</td>
<td>0.5</td>
<td>0.7397</td>
<td>0.8370</td>
<td>0.8713</td>
<td>0.8826</td>
<td>0.8862</td>
<td>0.8873</td>
<td>0.8877</td>
<td>0.8878</td>
</tr>
</tbody>
</table>

After 13 iterations, we have converged sufficiently close to the solution (fixed point) \( u^* = 0.887862 \) to have computed its value to 7 decimal places.

Remark: Inspection of the proof of Theorem 19.4 reveals that we never really used the differentiability of \( g \), except to verify the inequality

\[
|g(u) - g(v)| \leq \rho |u - v| \quad \text{for some fixed \( \rho \).} \tag{19.18}
\]

A function that satisfies (19.18) for all \( u \) nearby a given point \( v \) is called Lipschitz continuous, in honor of the 19th century German mathematician Rudolf Lipschitz. The Mean Value Theorem C.3 implies that any continuously differentiable function \( g \in C^1 \) is automatically Lipschitz continuous, but there are nondifferentiable examples. The simplest is the absolute value function \( g(u) = |u| \), which is Lipschitz continuous, since

\[
|g(u) - g(v)| = |u - v| \leq |u - v| \quad \text{for any } u, v \in \mathbb{R},
\]

but is not differentiable at \( u = 0 \). On the other hand, as its name indicates, Lipschitz continuity does imply continuity. Thus, stability of the fixed point follows from the weaker hypothesis that \( g(u) \) is Lipschitz continuous at \( u^* \) with Lipschitz constant \( \rho < 1 \).

Example 19.6. The simplest truly nonlinear example is a quadratic polynomial. The most important case is the so-called logistic map

\[
g(u) = \lambda u(1 - u), \tag{19.19}
\]

where \( \lambda \neq 0 \) is a fixed non-zero parameter. (The case \( \lambda = 0 \) is completely trivial. Why?) In fact, an elementary change of variables can make any quadratic iterative system into one involving a logistic map; see Exercise [ ].

The fixed points of the logistic map are the solutions to the quadratic equation

\[
u = \lambda u(1 - u), \quad \text{or} \quad \lambda u^2 - \lambda u + 1 = 0.
\]

Using the quadratic formula, we conclude that \( g(u) \) has two fixed points:

\[
u_1^* = 0, \quad \nu_2^* = 1 - \frac{1}{\lambda}.
\]

Let us apply Theorem 19.4 to determine their stability. The derivative is

\[
g'(u) = \lambda - 2 \lambda u, \quad \text{and so} \quad g'(\nu_1^*) = \lambda, \quad g'(\nu_2^*) = 2 - \lambda.
\]

Therefore, if \( |\lambda| < 1 \), the first fixed point is stable, while if \( 1 < \lambda < 3 \), the second fixed point is stable. For \( \lambda < -1 \) or \( \lambda > 3 \) neither fixed point is stable, and we expect the iterates to not converge at all.
Numerical experiments with this example show that it is the source of an amazingly
diverse range of behavior, depending upon the value of the parameter \( \lambda \). In the following
table, we display the results of iteration starting with initial point \( u^{(0)} = 1 \). As expected
from Theorem 19.4, the iterates converge to one of the fixed points in the range \(-1 < \lambda < 3\),
except when \( \lambda = 1 \). For \( \lambda \) a little bit larger than \( \lambda_1 = 3 \), the iterates do not converge to a
fixed point; an example appears in the table. But it does not take long for them to settle
down and switch back and forth between two particular values. This behavior indicates
that there is a (stable) period 2 orbit for the discrete dynamical system, in accordance
with the following definition.

**Definition 19.7.** A period \( k \) orbit of a discrete dynamical system is a solution that
satisfies \( u^{(n+k)} = u^{(n)} \) for all \( n = 0, 1, 2, \ldots \). The (minimal) period is the smallest positive
value of \( k \) for which this condition holds.

Thus, a fixed point
\[
u^{(0)} = u^{(1)} = u^{(2)} = \cdots
\]
is a period 1 orbit. A period 2 orbit satisfies
\[
u^{(0)} = u^{(2)} = u^{(4)} = \cdots \quad \text{and} \quad u^{(1)} = u^{(3)} = u^{(5)} = \cdots ,
\]
but \( u^{(0)} \neq u^{(1)} \), as otherwise the minimal period would be 1. Similarly, a period 3 orbit
has
\[
u^{(0)} = u^{(3)} = u^{(6)} = \cdots , \quad u^{(1)} = u^{(4)} = u^{(7)} = \cdots , \quad u^{(2)} = u^{(5)} = u^{(8)} = \cdots ,
\]
with \( u^{(0)}, u^{(1)}, u^{(2)} \) distinct. Stability implies that nearby iterates converge to this periodic
solution.

For the logistic map, the period 2 orbit persists until \( \lambda = \lambda_2 \approx 3.4495 \), after which
the iterates alternate between four values — a period 4 orbit. This again changes at
\( \lambda = \lambda_3 \approx 3.5441 \), after which the iterates end up alternating between eight values. In fact,
there is an increasing sequence of values
\[3 = \lambda_1 < \lambda_2 < \lambda_3 < \lambda_4 < \cdots \]
where, for any \( \lambda_n < \lambda \leq \lambda_{n+1} \), the iterates eventually follow a period \( 2^n \) orbit. Thus, as \( \lambda \) passes through each value \( \lambda_n \) the period of the orbit doubles from \( 2^n \) to \( 2 \cdot 2^n = 2^{n+1} \),
and the discrete dynamical system experiences a bifurcation. The bifurcation values \( \lambda_n \) lie
closer and closer together, piling up on an eventual limit \( \lambda_* = \lim_{n \to \infty} \lambda_n \approx 3.5699 \), at which
point the period has become infinitely large. The entire phenomena is known as a period
doubling cascade. Interestingly, the ratios of the distances between successive bifurcation
points approaches a well-defined limit,
\[
\frac{\lambda_{n+2} - \lambda_{n+1}}{\lambda_{n+1} - \lambda_n} \to 4.6692 \ldots , \tag{19.20}
\]
known as Feigenbaum’s constant. In the 1970’s, the American physicist Mitchell Feigen-
baum, [53], discovered that this period doubling cascade appears in a broad range of
discrete dynamical systems. Even more remarkably, in all cases, the corresponding ratios of distances between bifurcation points has the same limiting value. This was subsequently proved by Oscar Lanford in 1982, [101].

After $\lambda$ passes the limiting value $\lambda_*$, “all hell breaks loose”. The iterates become completely chaotic, moving at random over the interval $[0, 1]$. But this is not the end of the story. Embedded within this chaotic regime are certain small ranges of $\lambda$ where the system settles down to a stable orbit, whose period is not necessarily a power of 2. In fact, there exist values of $\lambda$ for which the iterates settle down to a stable orbit of period $m$ for any positive integer $m$. For instance, as $\lambda$ increases past $\lambda_3 \approx 3.83$, a period 3 orbit appears for a while; then it experiences a succession of period doubling cascade of period 6, 12, 24, ... orbits, each persisting on a shorter and shorter interval of parameter values, until chaos breaks out yet again. There is a well-prescribed order in which the periodic cases appear, and each period $m$ is followed by a very closely spaced sequence of period doubling bifurcations, of periods $2^n m$ for $n = 1, 2, 3, \ldots$, after which the iterates revert to completely chaotic behavior until the next periodic case emerges. The ratios of distances between bifurcation points have the same Feigenbaum limit (19.20). Finally, these periodic and chaotic windows all pile up on the ultimate parameter value $\lambda^* = 4$. And then, when $\lambda > 4$, all the iterates go off to $\infty$, and the system ceases to be interesting.

The reader is encouraged to write a simple computer program and perform some numerical experiments. In particular, Figure log shows the asymptotic behavior of the iterates for values of the parameter in the interesting range $2 < \lambda < 4$. The horizontal axis is $\lambda$, and the marked points show the ultimate fate of the iteration for the given value of $\lambda$. For instance, the single curve lying above low values of $\lambda$ represents a fixed point; this bifurcates into a pair of curves representing a stable period 2 orbit, which then bifurcates into 4 curves representing a period 4 orbit, and so on. Chaotic behavior is indicated by a somewhat random pattern of points lying above the value of $\lambda$. To plot this figure, we ran the iteration $u(n)$ for $0 \leq n \leq 100$, and then discarded the first 50 points, plotting the next 50 iterates $u(51), \ldots, u(100)$. Investigation of the fine detailed structure of the logistic map requires yet more iterations with increased accuracy. In addition one should discard more of the initial iterates so as to give the system enough time to settle down to a stable periodic orbit or continue in a chaotic manner.

Remark: So far, we have only looked at real scalar iterative systems. Complex discrete dynamical systems display yet more remarkable and fascinating behavior. The complex version of the logistic iteration equation leads to the justly famous Mandelbrot set, [102], with its stunning, psychedelic fractal structure, [120].

The rich range of phenomena in evidence even in such extremely simple nonlinear iterative systems is astounding. While intimations of this first appeared in the late nineteenth century research of the influential French mathematician Henri Poincaré, serious investigations were delayed until the advent of the computer era, which precipitated an explosion of research activity in the area of dynamical systems. Similar period doubling

\[\ ^\dagger \text{The term “chaotic” does have a precise mathematical definition, but the reader can take it more figuratively for the purposes of this elementary introduction.}\]
cascades and chaos can be found in a broad range of nonlinear systems, [7], and are often encountered in physical applications, [107]. A modern explanation of fluid turbulence is that it is a (very complicated) form of chaos.

**Quadratic Convergence**

Let us now return to the more mundane case when the iterates converge to a stable fixed point of the discrete dynamical system. In applications, we are interested in computing a precise\(^\dagger\) numerical value for the fixed point, and hence the speed of convergence of the iterates is of crucial importance.

According to Theorem 19.4, the convergence rate of an iterative system is essentially governed by the magnitude of the derivative \(|g'(u^*)|\) at the fixed point. The basic inequality (19.14) for the errors \(e^{(k)} = u^{(k)} - u^*\), namely

\[ |e^{(k+1)}| \leq \rho |e^{(k)}|, \]

is known as a *linear convergence estimate*. It means that the error decreases by a factor of at least \(\rho\) at each step. If the \(k^{th}\) iterate \(u^{(k)}\) approximates the fixed point \(u^*\) correctly to \(m\) decimal places, so \(|e^{(k)}| < 0.5 \times 10^{-m}\), then the \((k + 1)^{st}\) iterate satisfies

\[ |e^{(k+1)}| < 0.5 \times 10^{-m} \rho = 0.5 \times 10^{-m + \log_{10} \rho}. \]

More generally, for any \(j > 0\),

\[ |e^{(k+j)}| < 0.5 \times 10^{-m} \rho^j = 0.5 \times 10^{-m + j \log_{10} \rho}, \]

which means that the \((k + j)^{th}\) iterate \(u^{(k+j)}\) has at least\(^\ddagger\)

\[ m - j \log_{10} \rho = m + j \log_{10} \rho^{-1} \]

correct decimal places. For instance, if \(\rho = 0.1\) then each new iterate produces one new decimal place of accuracy (at least), while if \(\rho = 0.9\) then it typically takes \(22 \approx -1/\log_{10} 0.9\) iterates to produce just one additional accurate digit!

As a consequence, there is a huge advantage — particularly in the application of iterative methods to the numerical solution of equations — to arranging that \(|g'(u^*)|\) be as small as possible. The fastest convergence rate of all will occur when \(g'(u^*) = 0\). Now the constant \(\rho\) in (19.14) can be taken to be arbitrarily small, although the smaller \(\rho\) is, the smaller the interval \(|v - u^*| < \delta\) on which (19.14) applies, and so the closer one must be to the fixed point. Be that as it may, once the iterates start converging, they will get closer and closer to the fixed point, and so the rate of convergence will speed up accordingly. In fact, for such functions, the rate of convergence is not just slightly, but dramatically faster than linear.

\(^\dagger\) The degree of precision is to be specified by the user and the application.

\(^\ddagger\) Note that since \(\rho < 1\), the logarithm \(\log_{10} \rho^{-1} = -\log_{10} \rho > 0\) is positive.
Theorem 19.8. Let \( g(u) \in C^2 \). Suppose \( u^* = g(u^*) \) is a fixed point such that \( g'(u^*) = 0 \). Then, for all iterates \( u^{(k)} \) sufficiently close to \( u^* \), the errors \( e^{(k)} = u^{(k)} - u^* \) satisfy the quadratic convergence estimate

\[
|e^{(k+1)}| \leq \sigma |e^{(k)}|^2
\]

for some constant \( \sigma > 0 \).

**Proof:** Just as in the proof of the linear convergence estimate (19.14), the proof relies on approximating the function by a simpler function near the fixed point. For linear convergence, an affine approximation sufficed, but in this case we require a higher order, quadratic approximation. Instead of the mean value formula (19.12), we now use the first order Taylor expansion (C.6) of \( g \) near \( u^* \):

\[
g(u) = g(u^*) + g'(u^*)(u - u^*) + \frac{1}{2} g''(w)(u - u^*)^2,
\]

(19.22)

where the error term depends on an (unknown) point \( w \) that lies between \( u \) and \( u^* \). At a fixed point, the constant term is \( g(u^*) = u^* \). Furthermore, under our hypothesis \( g'(u^*) = 0 \), and so the Taylor expansion (19.22) reduces to

\[
g(u) - u^* = \frac{1}{2} g''(w)(u - u^*)^2.
\]

Therefore,

\[
|g(u) - u^*| \leq \sigma |u - u^*|^2,
\]

(19.23)

where \( \sigma \) is chosen so that

\[
\frac{1}{2} |g''(w)| \leq \sigma
\]

(19.24)

for all \( w \) sufficiently close to \( u^* \). Therefore, the magnitude of \( \sigma \) is governed by the size of the second derivative of the iterative function \( g(u) \) near the fixed point. We apply (19.23) to estimate the error

\[
|e^{(k+1)}| = |u^{(k+1)} - u^*| = |g(u^{(k)}) - g(u^*)| \leq \sigma |u^{(k)} - u^*|^2 = \sigma |e^{(k)}|^2,
\]

which establishes the quadratic convergence estimate (19.21).

Q.E.D.

Let us see how the quadratic estimate (19.21) speeds up the convergence rate. Following our earlier argument, suppose \( u^{(k)} \) is correct to \( m \) decimal places, so

\[
|e^{(k)}| < 0.5 \times 10^{-m}.
\]

Then (19.21) implies that

\[
|e^{(k+1)}| < 0.5 \times (10^{-m})^2 \sigma = 0.5 \times 10^{-2m+\log_{10} \sigma},
\]

and so \( u^{(k+1)} \) has \( 2m - \log_{10} \sigma \) accurate decimal places. If \( \sigma \approx g''(u^*) \) is of moderate size, we essentially *double* the number of accurate decimal places in just a single iterate! A second iteration will double the number of accurate digits yet again. Thus, the convergence of a quadratic iteration scheme is *extremely* rapid, and, barring round-off errors, one can produce any desired number of digits of accuracy in a very short time. For example, if we start with an initial guess that is accurate in the first decimal digit, then a linear iteration with \( \rho = 0.1 \) will require 49 iterations to obtain 50 decimal place accuracy, whereas a quadratic iteration (with \( \sigma = 1 \)) will only require 6 iterations to obtain \( 2^6 = 64 \) decimal places of accuracy!
Example 19.9. Consider the function

\[ g(u) = \frac{4u^3 + 2u - 1}{3u^2 + 1}. \]

There is a unique fixed point \( u^* = g(u^*) \) which is the solution to the cubic equation

\[ u^3 + u - 1 = 0. \]

Note that

\[ g'(u) = \frac{6u^4 + 6u^2 - 6u}{(3u^2 + 1)^2} = \frac{6u(u^3 + u - 1)}{(3u^2 + 1)^2}, \]

and hence \( g'(u^*) \) vanishes at the fixed point. Theorem 19.8 implies that the iterations should exhibit quadratic convergence to the root. Indeed, we find, starting with \( u^{(0)} = 0 \), the following values. Not the dramatically faster convergence, especially when contrasted with the linearly convergent scheme based on \( 1 \).

For a general discrete dynamical system, the appearance of a quadratically convergent fixed point is a matter of luck. The construction of general purpose quadratically convergent iterative methods for solving equations will be the focus of the following Section 19.2.

Vector-Valued Iteration

Extending the preceding analysis to vector-valued iterative systems is not especially difficult. We will build on our experience with linear iterative systems, and so the reader should review the basic concepts and results from Chapter 10 before proceeding to the nonlinear cases presented here.

We begin by fixing a norm \( \| \cdot \| \) on \( \mathbb{R}^n \). Since we will also be computing the associated matrix norm \( \| A \| \), as defined in Theorem 10.17, it may be computationally more convenient to adopt either the 1 or the \( \infty \) norms rather than the standard Euclidean norm. As far as the theory goes, however, the precise choice of norm is unimportant.

We begin by defining the vector-valued counterpart of the basic linear convergence condition (19.18).

Definition 19.10. A function \( g: \mathbb{R}^n \to \mathbb{R}^n \) is Lipschitz continuous at a point \( a \in \mathbb{R}^n \) if there exists a constant \( \rho \geq 0 \), known as the Lipschitz constant, such that

\[ \| g(u) - g(a) \| \leq \rho \| u - a \| \] (19.25)

for all \( u \) sufficiently close to \( a \), i.e., \( \| u - a \| < \delta \) for some fixed \( \delta > 0 \).

Example 19.11. Consider the function \( g(u) = \left( \frac{|u-v|}{\max\{|u|,|v|\}} \right) \), defined for \( u = (u,v)^T \in \mathbb{R}^2 \). Although \( g \) is not differentiable, it does satisfy the Lipschitz estimate (19.25) for the 1 norm \( \| u \|_1 = |u| + |v| \). Indeed,

\[ \| g(u) - g(a) \| \leq \left| |u - v| - |a - b| \right| + \left| \max\{|u|,|v|\} - \max\{|a|,|b|\} \right| \leq 2 \left( |u - a| + |v - b| \right) = 2 \| u - a \|_1. \]

Thus, (19.25) holds with uniform Lipschitz constant \( \rho = 2 \).
Remark: The notion of “Lipschitz continuity” appears to depend on the underlying choice of matrix norm. However, the fact that all norms on a finite-dimensional vector space are essentially equivalent — see Theorem 3.19 — implies that this concept is, in fact, independent of the choice of norm. However, one should keep in mind that the value of the Lipschitz constant $\lambda$ is norm-dependent.

The Lipschitz inequality (19.25) provides an immediate proof of the basic convergence theorem for iteration of a discrete dynamical system (19.1). Recall that a fixed point is called asymptotically stable if $u^{(k)} \to u^*$ for every initial condition $u^{(0)} = c$ sufficiently close to $u^*$.

**Theorem 19.12.** If $u^* = g(u^*)$ is a fixed point for the discrete dynamical system (19.1) and $g$ is Lipschitz continuous at $u^*$ with Lipschitz constant $\lambda < 1$, then $u^*$ is an asymptotically stable fixed point.

**Proof:** The proof is a copy of the last part of the proof of Theorem 19.4. We write

$$\|u^{(k+1)} - u^*\| = \|g(u^{(k)}) - g(u^*)\| \leq \lambda \|u^{(k)} - u^*\|,$$

using the assumed Lipschitz estimate (19.25). Iterating this basic inequality immediately demonstrates that

$$\|u^{(k)} - u^*\| \leq \lambda^k \|u^{(0)} - u^*\| \quad \text{for} \quad k = 0, 1, 2, 3, \ldots.$$

Since $\lambda < 1$, the right hand side tends to 0 as $k \to \infty$, and hence $u^{(k)} \to u^*$. Q.E.D.

For more complicated functions, the direct verification of the Lipschitz inequality (19.25) is not particularly easy. However, as in the scalar case, any continuously differentiable function is automatically Lipschitz continuous.

**Theorem 19.13.** If $g(u) \in C^1$ has continuous first order partial derivatives for all $u$ sufficiently close to $u^*$, then $g$ is Lipschitz continuous at $u^*$.

**Proof:** According to the first order Taylor expansion (C.10) of a vector-valued function at a point $u^*$ takes the form

$$g(u) = g(u^*) + g'(u^*)(u - u^*) + R(u - u^*). \quad (19.26)$$

Here

$$g'(u) = \begin{pmatrix}
\frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} & \cdots & \frac{\partial g_1}{\partial u_n} \\
\frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} & \cdots & \frac{\partial g_2}{\partial u_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial g_n}{\partial u_1} & \frac{\partial g_n}{\partial u_2} & \cdots & \frac{\partial g_n}{\partial u_n}
\end{pmatrix}, \quad (19.27)$$
is the $n \times n$ Jacobian matrix of the vector-valued function $\mathbf{g}$ whose entries are the partial derivatives of its individual components. The remainder term in (19.26) satisfies

$$\| R(\mathbf{v}) \| \leq \sigma \| \mathbf{v} \|^2 \quad \text{whenever} \quad \| \mathbf{v} \| \leq \varepsilon,$$

for some positive constant $\sigma > 0$. If the corresponding matrix norm of the Jacobian matrix at $\mathbf{u}^*$ satisfies

$$\| \mathbf{g}'(\mathbf{u}^*) \| = \rho^*,$$

then, by the triangle inequality and the definition (10.23) of matrix norm,

$$\| \mathbf{g}(\mathbf{u}) - \mathbf{g}(\mathbf{u}^*) \| = \| \mathbf{g}'(\mathbf{u}^*) (\mathbf{u} - \mathbf{u}^*) + R(\mathbf{u} - \mathbf{u}^*) \| \leq \| \mathbf{g}'(\mathbf{u}^*) (\mathbf{u} - \mathbf{u}^*) \| + \| R(\mathbf{u} - \mathbf{u}^*) \| \leq \| \mathbf{g}'(\mathbf{u}^*) \| \| \mathbf{u} - \mathbf{u}^* \| + \sigma \| \mathbf{u} - \mathbf{u}^* \|^2 \leq (\rho^* + \varepsilon \sigma) \| \mathbf{u} - \mathbf{u}^* \|, \quad (19.28)$$

whenever $\| \mathbf{u} - \mathbf{u}^* \| \leq \varepsilon$. This proves that $\mathbf{g}$ is Lipschitz continuous at $\mathbf{u}^*$ with Lipschitz constant $\rho = \rho^* + \varepsilon \sigma$. Note that, by choosing $\varepsilon$ small enough, we can ensure that the Lipschitz constant $\rho$ is arbitrarily close to the matrix norm $\rho^*$.

Q.E.D.

For a continuously differentiable function, then, asymptotic stability is a consequence of the size, or, more correctly, the spectral radius of the Jacobian matrix at the fixed point.

**Theorem 19.14.** Suppose $\mathbf{g}(\mathbf{u}) \in C^2$. If $\mathbf{u}^* = \mathbf{g}(\mathbf{u}^*)$ is a fixed point such that $\mathbf{g}'(\mathbf{u}^*)$ is a convergent matrix, then $\mathbf{u}^*$ is asymptotically stable. The rate of convergence of the iterative scheme $\mathbf{u}^{(k+1)} = \mathbf{g}(\mathbf{u}^{(k)})$ to $\mathbf{u}^*$ is governed by the spectral radius of $\mathbf{g}'(\mathbf{u}^*)$.

**Proof:** If $\mathbf{g}'(\mathbf{u}^*)$ is convergent, and hence has spectral radius strictly less than 1, then Corollary 10.29 assures us that there exists a matrix norm such that

$$\| \mathbf{g}'(\mathbf{u}^*) \| = \rho^* < 1. \quad (19.29)$$

Defining $\sigma$ as in the proof of Theorem 19.13, we then choose $\varepsilon > 0$ so that

$$\rho = \rho^* + \varepsilon \sigma < 1.$$

Then (19.28) implies that

$$\| \mathbf{g}(\mathbf{u}) - \mathbf{g}(\mathbf{u}^*) \| \leq \rho \| \mathbf{u} - \mathbf{u}^* \|, \quad \text{provided} \quad \| \mathbf{u} - \mathbf{u}^* \| < \varepsilon.$$

As before, this suffices to proves convergence of the iterates to $\mathbf{u}^*$. Q.E.D.

**Example 19.15.**

Theorem 19.14 tells us that initial values $\mathbf{u}^{(0)}$ that are sufficiently near a stable fixed point $\mathbf{u}^*$ are guaranteed to converge to it. In the linear case, closeness of the initial data to the fixed point was not, in fact, an issue; all stable fixed points are, in fact, globally stable. For nonlinear iteration, it is of critical importance, and one does not typically expect iteration starting with far away initial data to converge to the desired fixed point.

---

\[ \footnote{\text{We can use any convenient norm on } \mathbb{R}^n.} \]
An interesting (and difficult) problem is to determine the so-called basin of attraction of a stable fixed point, defined as the set of all initial data that ends up converging to it. As in the elementary logistic map (19.19), initial values that lie outside a basin of attraction can lead to divergent iterates, periodic orbits, or even exhibit chaotic behavior. The full range of possible phenomena is a subject of contemporary research in dynamical systems theory and in numerical analysis, [7].

The smaller the spectral radius or matrix norm of the Jacobian matrix at the fixed point, the faster the iterates converge to it. As in the scalar case, quadratic convergence will occur when the Jacobian matrix \( g'(u^*) = 0 \) is the zero matrix\(^{\dagger} \), i.e., all first order partial derivatives of the components of \( g \) vanish at the fixed point. The quadratic convergence estimate

\[
\| u^{(k+1)} - u^* \| \leq \sigma \| u^{(k)} - u^* \|^2
\]

(19.30)

is a consequence of the second order Taylor expansion at the fixed point. Details of the proof are left as an exercise.

**Example 19.16.**

In general, the existence of a fixed point of an iterative system is not automatic. One way is to observe the iterates starting with suitably selected initial data; if they converge, then Proposition 19.2 assures us that their limit is a fixed point. There is one important class of maps for which we have a theoretical justification, not only of the existence, but also the uniqueness of a fixed point.

**Definition 19.17.** A map \( g: \Omega \to \Omega \) is called a contraction mapping if it has Lipschitz constant \( \rho < 1 \) at all points in \( \Omega \).

Therefore, applying a contraction mapping reduces the distance between points. As a result, a contraction mapping shrinks the size of its domain; see Figure contract\(^{\dagger} \). As a result, as the iterations proceed, the domain gets smaller and smaller and the iterates become trapped. If the original domain is closed and bounded, then it is forced to shrink down to a single point, which is the unique fixed point of the iterative system.

The simplest example of a contraction mapping is the scaling map \( g(u) = \rho u \) with \( 0 < \rho < 1 \). Starting with the unit ball \( B_1 = \{ \| u \| \leq 1 \} \), at the \( k \)th iteration the points have been mapped into a contracted sphere of radius \( \rho^k \). As \( k \to \infty \) these contracted domains become smaller and smaller, converging in the limit to the unique fixed point \( u^* = 0 \). A precise statement of the Contraction Mapping Theorem follows; see [map] for the proof.

**Theorem 19.18.** If \( g: \Omega \to \Omega \) is a contraction mapping defined on a closed bounded domain \( \Omega \subset \mathbb{R}^n \) then \( g \) admits a unique fixed point \( u^* \in \Omega \). Moreover, starting with any initial point \( u^{(0)} \in \Omega \), the iterates necessarily converge to the fixed point \( u^{(k)} \to u^* \).

More sophisticated, powerful fixed point theorems require advanced knowledge of algebraic topology and will not be developed in this text. See [fixed] for details.

\(^{\dagger} \) Having zero spectral radius is not sufficient for quadratic convergence; see Exercise \(^{\dagger} \).

The solution of nonlinear equations and systems of equations is, of course, a problem of utmost importance in mathematics and its manifold applications. In the general situation, we are given a collection of $m$ functions depending upon $n$ variables, and we are interested in finding all solutions $\mathbf{u} = (u_1, u_2, \ldots, u_n)^T$ to the system

$$f_1(u_1, \ldots, u_n) = 0, \quad \cdots \quad f_m(u_1, \ldots, u_n) = 0. \quad (19.31)$$

In practice, as in the linear case, we are primarily interested in the case when the number of equations is equal to the number of unknowns, $m = n$, as one can only expect both existence and uniqueness of solutions in such situations. This point will be discussed in further detail below.

There is no universal direct solution method for nonlinear equations and systems comparable to Gaussian elimination. As a result, numerical solution techniques rely almost exclusively on iterative algorithms. In this section, we shall present the principal methods for numerically approximating the solution(s) to a system. We shall only discuss general purpose algorithms. Specialized methods for particular classes of equations, e.g., methods designed for solving polynomial equations, can be found in numerical analysis texts, e.g., [30, 121]. Of course, the most important specialized methods — those designed for solving linear systems — will continue to play a critical role, even in the nonlinear regime.

The Bisection Method

We begin, as always, with the scalar case. Thus, we are given a real-valued function $f: \mathbb{R} \to \mathbb{R}$, and seek its roots, i.e., the real† solution(s) to the scalar equation

$$f(u) = 0. \quad (19.32)$$

Here are some prototypical examples:

(a) Find the roots of the quintic polynomial equation

$$u^5 + u + 1 = 0. \quad (19.33)$$

Graphing the left hand side of the equation, as in Figure 19.1, convinces us that there is just one real root, lying somewhere between $-1$ and $-0.5$. While there are explicit algebraic formulas for the roots of quadratic, cubic, and quartic polynomials, a famous theorem‡ due to the Norwegian mathematician Nils Henrik Abel in the early 1800’s states that there is no such formula for generic fifth order polynomial equations.

(b) As noted in Example 19.5, the trigonometric Kepler equation

$$u - \epsilon \sin u = m$$

† Complex roots to complex equations will be discussed later.
‡ A modern proof of this fact relies on Galois theory, [62].
arises in the study of planetary motion. Here \( \epsilon, m \) are fixed constants, and we seek a corresponding solution \( u \). We have already looked at one iterative solution method for this equation.

(c) Chemistry

The most primitive method for solving scalar equations, and the only one that is guaranteed to work in all cases, is the bisection algorithm. While it has an iterative flavor, it cannot be properly classed as a method governed by functional iteration as defined in the preceding section, and so must be studied directly in its own right.

The starting point is the Intermediate Value Theorem, which we state in simplified form, without proof. See Figure 19.2 for an illustration, and [9] for a proof.

**Theorem 19.19.** Let \( f(u) \) be a continuous scalar function. Suppose we can find two points \( a < b \) where the values of \( f(a) \) and \( f(b) \) take opposite signs, so either \( f(a) < 0 \) and \( f(b) > 0 \), or \( f(a) > 0 \) and \( f(b) < 0 \). Then there exists at least one point \( a < u^* < b \) where \( f(u^*) = 0 \).

The hypothesis can be compactly written as \( f(a) f(b) < 0 \). Note that if \( f(a) = 0 \) or \( f(b) = 0 \), then finding a root is trivial. If \( f(a) \) and \( f(b) \) have the same sign, then there may or may not be a root in between. Figure 19.3 plots the functions \( u^2 + 1 \), \( u^2 \) and \( u^2 - 1 \), on the interval \(-2 \leq u \leq 2\). The first has two simple roots; the second has a single double root, while the third has no root. Also, continuity of the function on the entire interval \([a, b]\) is an essential hypothesis. For example, the function \( f(u) = 1/u \) satisfies \( f(-1) = -1 \) and \( f(1) = 1 \), but there is no root to the equation \( 1/u = 0 \).

Note carefully that the Theorem 19.19 does not say there is a unique root between \( a \) and \( b \). There may be many roots, or even, in pathological examples, infinitely many. All the theorem guarantees is that there is at least one root.
Once we are assured that a root exists, bisection amounts to a “divide and conquer” strategy. Starting with the endpoints, the goal is to locate a root \( a < u^* < b \) between them. Lacking any additional evidence, a good strategy would be to try the midpoint \( c = \frac{1}{2}(a + b) \) as a first guess for the root. If, by some miracle, \( f(c) = 0 \), then we are done, since we have found a solution! Otherwise (and typically) we look at the sign of \( f(c) \). There are two possibilities. If \( f(a) \) and \( f(c) \) are of opposite signs, then the Intermediate Value Theorem tells us that there is a root \( u^* \) lying between \( a < u^* < c \). Otherwise, \( f(c) \) and \( f(b) \) must have opposite signs, and so there is a root \( c < u^* < b \). In either event, we apply the same method to the interval in which we are assured a root lies, and repeat the procedure. Each iteration halves the length of the interval, and chooses the half in which a root must be. (There may, of course, be a root in the other half, but we cannot be sure of this, and so discard it from further consideration.) The root we home in on lies trapped in intervals of smaller and smaller width, and so convergence of the method is guaranteed. Figure bisect illustrates the steps in a particular example.
Example 19.20. The roots of the quadratic equation

\[ f(u) = u^2 + u - 3 = 0 \]

are given by the quadratic formula

\[ u_1^* = \frac{-1 + \sqrt{13}}{2} \approx 1.302775 \ldots, \quad u_2^* = \frac{-1 - \sqrt{13}}{2} \approx -2.302775 \ldots. \]

Let us see how one might approximate them by applying the Bisection Algorithm. We start the procedure by choosing the points \( a = u^{(0)} = 1, \ b = v^{(0)} = 2, \) noting that \( f(1) = -1 \) and \( f(2) = 3 \) have opposite signs and hence we are guaranteed that there is at least one root between 1 and 2. In the first step we look at the midpoint of the interval \([1, 2]\), which is 1.5, and evaluate \( f(1.5) = .75 \). Since \( f(1) = -1 \) and \( f(1.5) = .75 \) have opposite signs, we know that there is a root lying between 1 and 1.5. Thus, we use \( u^{(1)} = 1 \) and \( v^{(1)} = 1.5 \) as the endpoints of the next interval, and continue. The next midpoint is at 1.25, where \( f(1.25) = -.1875 \) has the opposite sign to \( f(1.5) = .75 \), and so a root lies between \( u^{(2)} = 1.25 \) and \( v^{(2)} = 1.5 \). The process is then iterated as long as desired — or, more practically, as long as your computer’s precision does not become an issue.

The accompanying table displays the result of the algorithm, rounded off to four decimal digits. Thus, after 14 iterations the Bisection Algorithm has computed the positive root \( u_1^* \) correctly to 4 decimal places. A similar bisection starting with the interval from \( u^{(1)} = -3 \) to \( v^{(1)} = -2 \) will produce the negative root.
The formal implementation of the algorithm is governed by the following program. The endpoints of the $k$th interval are denoted by $u^{(k)}$ and $v^{(k)}$. The midpoint is $w^{(k)} = \frac{1}{2}(u^{(k)}+v^{(k)})$, and the main decision is whether $w^{(k)}$ should be the right or left hand endpoint of the new interval. The integer $n$, governing the number of iterations, will be prescribed in accordance with how close we wish to approximate the solution $u^*$. 

The algorithm produces two sequences of approximations such that $u^{(k)} < u^* < v^{(k)}$ lies between them. Both converge monotonically to the root, one from below and the other from above:

$$a = u^{(0)} \leq u^{(1)} \leq u^{(2)} \leq \cdots \leq u^{(k)} \rightarrow u^* \leftarrow v^{(k)} \leq \cdots \leq v^{(2)} \leq v^{(1)} \leq v^{(0)} = b.$$ 

In other words, the solution $u^*$ is trapped inside a sequence of intervals $[u^{(k)}, v^{(k)}]$ of progressively shorter and shorter length. Since we cut the interval in half at each step of the algorithm, the length of the interval $[u^{(k)}, v^{(k)}]$ is exactly half that of $[u^{(k-1)}, v^{(k-1)}]$, and so

$$v^{(k)} - u^{(k)} = \frac{1}{2}(v^{(k-1)} - u^{(k-1)}).$$

Iterating this formula, we conclude that

$$v^{(n)} - u^{(n)} = \left(\frac{1}{2}\right)^n(v^{(0)} - u^{(0)}) = \left(\frac{1}{2}\right)^n(b - a).$$

The final approximation

$$w^{(n)} = \frac{1}{2}(u^{(n)} + v^{(n)})$$

lies in middle of its interval, and hence must be within a distance

$$|w^{(n)} - u^*| \leq \frac{1}{2}(v^{(n)} - u^{(n)}) = \left(\frac{1}{2}\right)^{n+1}(b - a)$$

of the root. Consequently, if we want to approximate the root within a prescribed tolerance $\varepsilon$, we should choose the number of iterations $n$ so that

$$\left(\frac{1}{2}\right)^{n+1}(b - a) < \varepsilon, \quad \text{or} \quad n > \log_2\frac{b - a}{\varepsilon} - 1.$$

(19.34)
The Bisection Method

start
  if \( f(a) f(b) < 0 \) set \( u^{(0)} = a, \ v^{(0)} = b \)
  for \( k = 0 \) to \( n - 1 \)
    set \( w^{(k)} = \frac{1}{2}(u^{(k)} + v^{(k)}) \)
    if \( f(w^{(k)}) = 0 \), stop; print \( u^* = w^{(k)} \)
    if \( f(w^{(k)}) f(u^{(k)}) < 0 \), set \( u^{(k+1)} = w^{(k)}, \ v^{(k+1)} = v^{(k)} \)
    else set \( u^{(k+1)} = u^{(k)}, \ v^{(k+1)} = w^{(k)} \)
  next \( k \)
  print \( u^* = w^{(n)} = \frac{1}{2}(u^{(n)} + v^{(n)}) \)
end

Theorem 19.21. If \( f(u) \) is a continuous function, with \( f(a) f(b) < 0 \), then the bisection algorithm starting with \( u^{(0)} = a, v^{(0)} = b \) will converge to a solution to \( f(u) = 0 \) lying between \( a \) and \( b \). After \( n \) steps, the midpoint \( w^{(n)} = \frac{1}{2}(u^{(n)} + v^{(n)}) \) will be within a tolerance of \( \varepsilon = 2^{-n-1}(b - a) \) of the solution.

For example, in the case of the quadratic equation in Example 19.20, after 14 iterations, we have approximated the positive root to within

\[
\varepsilon = \left(\frac{1}{2}\right)^{15} (2 - 1) \approx 3.052 \times 10^{-5},
\]

reconfirming our observation that we have accurately computed the first four decimal places of the root. If we need 10 decimal places, we set our tolerance to \( \varepsilon = 10^{-11} \), and so, according to (19.34), must perform \( n = 36 > 35.54 \approx \log_2 10^{11} - 1 \) successive bisections.

Example 19.22. As noted at the beginning of this section, the quintic equation

\[
f(u) = u^5 + u + 1 = 0
\]

has one real root, whose value can be readily computed by bisection. We start the algorithm with the initial points \( u^{(0)} = -1, v^{(0)} = -0.5 \), noting that \( f(-1) = -1 < 0 \) while \( f(0) = 1 > 0 \) are of opposite signs. In order to compute the root to 6 decimal places, we set \( \varepsilon = 10^{-7} \) in (19.34), and so need to perform \( n = 23 > 22.25 \approx \log_2 10^7 - 1 \) bisections. Indeed, the algorithm produces the approximation \( u^* \approx -0.754878 \) to the root, and the displayed digits are guaranteed to be accurate.

Fixed Point Methods

The Bisection method converges in all cases — provided it can be properly started by locating two points where the function takes opposite signs. This may be tricky if the function has two very closely spaced roots and is, say, negative only for a very small
interval between them, and may be impossible for multiple roots, e.g., the root $u^* = 0$ of the quadratic function $f(u) = u^2$. When applicable, its convergence rate is completely predictable, but not especially fast. Worse, it has no immediately apparent extension to systems of equations, since there is no counterpart to the Intermediate Value Theorem for vector-valued functions.

Most other methods for solving equations rely on some form of fixed point iteration. Thus, we seek to replace the system of equations (19.32) with a fixed point system

$$u = g(u). \quad (19.35)$$

The key requirements are

(a) The solution $u^*$ to (19.32) is also a fixed point for equation (19.35), and

(b) $u^*$ is, in fact a stable fixed point, so the Jacobian matrix $g'(u^*)$ is a convergent matrix, or, slightly more restrictively, $\|g'(u^*)\| < 1$ for a given matrix norm.

If both requirements are satisfied, then, provided we choose the initial iterate $u^{(0)} = c$ sufficiently close to $u^*$, the iterates $u^{(k)}$ will converge to the desired solution $u^*$ as $k \to \infty$. Thus, the key to the practical use of functional iteration for solving equations is the proper design of an iterative system, coupled with a reasonably good initial guess for the solution.

Example 19.23. To solve the cubic equation

$$f(u) = u^3 - u - 1 = 0 \quad (19.36)$$

we note that $f(1) = -1$ while $f(2) = 5$, and so there is a root between 1 and 2. Indeed, the bisection algorithm gives the approximate value $u^* \approx 1.3247$ after 17 iterations.

Let us try to find the same root by fixed point iteration. As a first, naïve, guess, we rewrite the cubic equation in fixed point form

$$u = 1 - u^3 = \tilde{g}(u).$$

Starting with the initial guess $u^{(0)} = 1.5$, the successive iterates are given by

$$u^{(k+1)} = \tilde{g}(u^{(k)}) = 1 - (u^{(k)})^3, \quad k = 0, 1, 2, \ldots .$$

However, their values

$$u^{(0)} = 1.5, \quad u^{(1)} = -2.375, \quad u^{(2)} = 14.3965,$$

$$u^{(3)} = -2.983, \quad u^{(4)} = 2.654 \times 10^{10}, \quad u^{(5)} = -1.869 \times 10^{31}, \quad \ldots$$

rapidly become unbounded and fail to converge. This could have been predicted by the convergence criterion in Theorem 19.4. Indeed, $\tilde{g}'(u) = -3u^2$ and so $|\tilde{g}'(u)| > 3$ for all $1 \leq u$, including the root $u^*$. This means that $u^*$ is an unstable fixed point, and we cannot expect the iterates to converge to it.

On the other hand, we can rewrite the equation (19.36) in the alternative iterative form

$$u = \sqrt[3]{1 + u} = g(u).$$

In this case

$$0 \leq g'(u) = \frac{1}{3(1 + u)^{2/3}} \leq \frac{1}{3} \quad \text{for} \quad u > 0.$$
Thus, the stability condition (19.13) is satisfied, and we anticipate convergence at a rate of at least $\frac{1}{3}$. (The bisection method converges more slowly, at rate $\frac{1}{2}$.) Indeed, the first few iterates $u^{(k+1)} = \sqrt{1 + u^{(k)}}$ are
\[ 1.5, \quad 1.3571, \quad 1.33086, \quad 1.32588, \quad 1.32494, \quad 1.32476, \quad 1.32473, \]
and we have converged to the root, correct to four decimal places, in only 6 iterations.

**Newton’s Method**

As we learned in Section 19.1, the speed of convergence of an iterative method based on a scalar function $g(u)$ is governed by the magnitude of its derivative, $|g'(u^*)|$, at the fixed point. Thus, to design an iterative method to solve an equation $f(u) = 0$, we need

(a) a function $g(u)$ whose fixed points $u^*$ coincide with the solutions,

(b) whose derivative at the fixed point is as small as possible.

In particular, if we can arrange that $g'(u^*) = 0$, then, instead of a relatively slow linear convergence rate, the numerical solution method will satisfy the dramatically faster quadratic convergence estimate of Theorem 19.8, with all its consequent advantages.

Now, the first condition requires that $g(u) = u$ whenever $f(u) = 0$. A little thought will convince you that the iterative function should take the form

$$ g(u) = u - \lambda(u) f(u), \quad (19.37) $$

where $\lambda(u)$ is a reasonably nice function. If $f(u^*) = 0$, then clearly $u^* = g(u^*)$, and so $u^*$ is a fixed point. The converse holds provided $\lambda(u) \neq 0$ is never zero.

For a quadratically convergent method, the second requirement is that the derivative of $g(u)$ be zero at the fixed point solutions. We compute

$$ g'(u) = 1 - \lambda'(u) f(u) - \lambda(u) f'(u). $$

Thus, $g'(u^*) = 0$ at a solution to $f(u^*) = 0$ if and only if

$$ 0 = 1 - \lambda'(u^*) f(u^*) - \lambda(u^*) f'(u^*) = 1 - \lambda(u^*) f'(u^*). $$

Consequently, we should require that

$$ \lambda(u^*) = \frac{1}{f'(u^*)}, \quad (19.38) $$

to ensure a quadratically convergent iterative scheme. This assumes that $f'(u^*) \neq 0$, which means that $u^*$ is a simple root of $f$. We leave aside multiple roots, which require a different argument and method, to be outlined in Exercise [1].

Of course, there are many functions $\lambda(u)$ that satisfy (19.38), since we only need to specify its value at a single point. The problem is that we do not know $u^*$ — after all this is what we are trying to compute — and so cannot compute the value of the derivative of $f$ there. However, we can circumvent this apparent difficulty by a simple device: we impose equation (19.38) at all points,

$$ \lambda(u) = \frac{1}{f'(u)}, \quad (19.39) $$

2/25/04 847 © 2004 Peter J. Olver
which certainly guarantees that it hold at the solution \( u^* \). The result is the function

\[
g(u) = u - \frac{f(u)}{f'(u)},
\]  

(19.40)

that yields the iteration scheme known as Newton’s method. It dates back to Isaac Newton, the founder of the calculus, and, to this day, remains the most important general purpose algorithm for solving equations. Newton’s method starts with an initial guess \( u^{(0)} \) to be supplied by the user, and then successively computes

\[
u^{(k+1)} = u^{(k)} - \frac{f(u^{(k)})}{f'(u^{(k)})}.
\]  

(19.41)

Provided the initial guess is sufficiently close, the iterates \( u^{(k)} \) are guaranteed to converge to the (simple) root \( u^* \) of \( f \).

**Theorem 19.24.** Suppose \( f(u) \in C^2 \) is twice continuously differentiable. Let \( u^* \) be a solution to the equation \( f(u) = 0 \) such that \( f'(u^*) \neq 0 \). Given an initial guess \( u^{(0)} \) sufficiently close to \( u^* \), the Newton iteration scheme (19.41) converges at a quadratic rate to the solution \( u^* \).

**Proof:** By continuity, if \( f'(u^*) \neq 0 \), then \( f'(u) \neq 0 \), and hence the Newton iterative function (19.40) is well defined and continuously differentiable for all \( u \) sufficiently close to \( u^* \). Since \( g'(u) = f(u) f''(u)/f'(u)^2 \), we have \( g'(u^*) = 0 \), as promised by our construction. Hence, the result is an immediate consequence of Theorem 19.8. \( Q.E.D. \)

**Example 19.25.** Consider the cubic equation

\[
f(u) = u^3 - u - 1 = 0,
\]

that we already solved in Example 19.23. The function used in the Newton iteration is

\[
g(u) = u - \frac{f(u)}{f'(u)} = u - \frac{u^3 - u - 1}{3u^2 - 1},
\]

which is well-defined as long as \( u \neq \pm \frac{1}{\sqrt{3}} \). We will try to avoid these singular points. The iterative procedure

\[
u^{(k+1)} = g(u^{(k)}) = u^{(k)} - \frac{(u^{(k)})^3 - u^{(k)} - 1}{3(u^{(k)})^2 - 1}
\]

with initial guess \( u^{(0)} = 1.5 \) produces the following values:

1.5, 1.34783, 1.32520, 1.32472,

which gives the root correctly to 5 decimal places after only three iterations. The quadratic convergence of Newton’s method implies that, roughly, each new iterate doubles the number of correct decimal places. Thus, if we need to compute the root accurately to 40
The function \( f(u) = u^3 - \frac{3}{2} u^2 + \frac{5}{9} u - \frac{1}{27} \).

It takes sixteen iterations of the bisection algorithm starting with the three subintervals \([0, \frac{1}{3}]\), \([\frac{1}{3}, \frac{2}{3}]\) and \([\frac{2}{3}, 1]\) to produce the roots to six decimal places:

\[
u_1^* \approx .085119, \quad u_2^* \approx .451805, \quad u_3^* \approx .963076.
\]

Incidentally, if we start with the interval \([0, 1]\) and apply bisection, we converge (perhaps surprisingly) to the largest root \( u_3^* \) in 17 iterations.

Fixed point iteration based on the formulation

\[
u = g(u) = -u^3 + \frac{3}{2} u^2 + \frac{4}{9} u + \frac{1}{27}
\]

\(\dagger\) This assumes we are working in a sufficiently high precision arithmetic so as to avoid round-off errors.
can be used to find the first and third roots, but not the second root. For instance, starting with \( u^{(0)} = 0 \) produces \( u_1^* \) to 5 decimal places after 23 iterations, whereas starting with \( u^{(0)} = 1 \) produces \( u_3^* \) to 5 decimal places after 14 iterations. The reason we cannot produce \( u_2^* \) is due to the magnitude of the derivative 
\[
g'(u) = -3u^2 + 3u + \frac{4}{9}
\]
at the roots, which is
\[
g'(u_1^*) \approx 0.678065, \quad g'(u_2^*) \approx 1.18748, \quad g'(u_3^*) \approx 0.551126.
\]
Thus, \( u_1^* \) and \( u_3^* \) are stable fixed points, but \( u_2^* \) is unstable. However, because \( g'(u_1^*) \) and \( g'(u_3^*) \) are both bigger than 0.5, this iterative algorithm converges slower than ordinary bisection!

Finally, Newton’s method is based upon iteration of the function
\[
g(u) = u - \frac{f(u)}{f'(u)} = u - \frac{u^3 - \frac{3}{2}u^2 + \frac{5}{9}u - \frac{1}{27}}{3u^2 - 3u + \frac{2}{9}}.
\]
Starting with an initial guess of \( u^{(0)} = 0 \), the method computes \( u_1^* \) to 5 decimal places after only 4 iterations; starting with \( u^{(0)} = .5 \), it produces \( u_2^* \) after 2 iterations; while starting with \( u^{(0)} = 1 \) produces \( u_3^* \) after 3 iterations — a dramatic speed up over the other two methods.

Newton’s method has a very pretty graphical interpretation, that helps us understand what is going on and why it converges so fast. Given the equation \( f(u) = 0 \), suppose we know an approximate value \( u = u^{(k)} \) for a solution. Nearby \( u^{(k)} \), we can approximate the nonlinear function \( f(u) \) by its tangent line at the given point \( u^{(k)} \), which has the equation
\[
y = f(u^{(k)}) + f'(u^{(k)})(u - u^{(k)}). \tag{19.42}
\]
As long as the tangent line is not horizontal — which requires \( f'(u^{(k)}) \neq 0 \) — it crosses the axis at the abscissa
\[
u^{(k+1)} = u^{(k)} - \frac{f(u^{(k)})}{f'(u^{(k)})},
\]
which represents a new, and, presumably more accurate, approximation to the desired root. The procedure is illustrated pictorially in Figure Newton. Note that the passage from \( u^{(k)} \) to \( u^{(k+1)} \) is exactly the Newton iteration step (19.41). In this manner, Newton’s method can be viewed as successive approximation of the function by its tangent line and then using the root of the resulting affine function as the next approximation to the root of the function.

Given sufficiently accurate initial guesses, Newton’s method will then rapidly produce accurate values for the simple roots to the equation in question. In practice, barring special structure in the problem, Newton’s method is the root-finding algorithm of choice. The one caveat is that we need to come up with a reasonably close initial guess to the root we are seeking. Otherwise, there is no guarantee that it will converge at all, although if the Newton iterations do converge, we know that the limiting value is a root of our equation.
The behavior of Newton’s method as we change parameters and vary the initial guess is very similar to the logistic map, and includes period doubling bifurcations and chaotic behavior. The reader is invited to experiment with simple examples, some of which are provided in Exercise 1. For further details, see [120].

**Example 19.27.** For fixed values of the eccentricity $\epsilon$, Kepler’s equation

$$u - \epsilon \sin u = m$$

(19.43)
can be viewed as a implicit equation defining the eccentric anomaly $u$ as a function of the mean anomaly $m$. To solve the equation by Newton’s method, we introduce the iterative function

$$g(u) = u - \frac{u - \epsilon \sin u - m}{1 - \epsilon \cos u}.$$  

Notice that when $|\epsilon| < 1$, the denominator never vanishes and so the iteration remains well-defined everywhere. Starting with an initial guess $u^{(0)}$, we are assured that the method will quickly converge to the solution.

Fixing the eccentricity $\epsilon$, we can employ a **continuation method** to determine how the solution $u^* = h(m)$ depends upon the mean anomaly $m$. Namely, we start at $m = m_0 = 0$ with the obvious solution $u^* = h(0) = 0$. Then, to compute the solution at successive closely spaced values $0 < m_1 < m_2 < m_3 < \cdots$, we use the previously computed value as an initial guess $u^{(0)} = h(m_k)$ for the value of the solution at the next mesh point $m_{k+1}$, and run the Newton scheme until it converges to the value $u^* = h(m_{k+1})$. As long as $m_{k+1}$ is reasonably close to $m_k$, Newton’s method will converge to the solution quite quickly.

The continuation method will quickly produce the values of $u$ at the sample points $m_k$. Intermediate values can either be determined by an interpolation scheme, e.g., a cubic spline fit of the data, or by running the Newton scheme using the closest known value as an initial condition. A plot for the value $\epsilon = .5$ appears in Figure kepler1.

**Systems of Equations**

Let us now turn our attention to systems of equations. We shall only consider the case when there are the same number of equations as unknowns:

$$f_1(u_1, \ldots, u_n) = 0, \quad \ldots \quad f_n(u_1, \ldots, u_n) = 0.$$  

(19.44)

We shall write the system (19.44) in vector form

$$f(u) = 0,$$ 

(19.45)

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a vector-valued function of $n$ variables. Also, we do not necessarily require that $f$ be defined on all of $\mathbb{R}^n$, although this does simplify the exposition to a certain degree.

We shall only consider solutions that are isolated, meaning separated from all the others. More formally:

**Definition 19.28.** A solution $u^*$ to a system $f(u) = 0$ is called **isolated** if there exists $\delta > 0$ such that $f(u) \neq 0$ for all $u$ satisfying $0 < \|u - u^*\| < \delta$.  


2/25/04 851 © 2004 Peter J. Olver
Example 19.29. Consider the planar equation

\[ x^2 + y^2 = (x^2 + y^2)^2. \]

Rewriting the equation in polar coordinates as

\[ r = r^2 \quad \text{or} \quad r(r - 1) = 0, \]

we immediately see that the solutions consist of the origin \( x = y = 0 \) and all points on the unit circle \( r^2 = x^2 + y^2 = 1 \). Only the origin is an isolated solution.

Typically, the solutions to a system of \( n \) equations in \( n \) unknowns are isolated, although this is not always the case. For example, if \( A \) is a singular \( n \times n \) matrix, then the solutions to \( Au = 0 \) consist of a nontrivial subspace of \( \mathbb{R}^n \) and so are not isolated. Nonlinear systems with non-isolated solutions can similarly be viewed as having some form of degeneracy. In general, the computation of non-isolated solutions, e.g., solving the implicit equations for a curve or surface, is a much more difficult problem, and we will not attempt to discuss these issues in this introductory presentation. However, our continuation approach to the Kepler equation in Example 19.27 gives a hint as to how one might proceed in such situations.

In the case of a single scalar equation, the simple roots are the most amenable to practical computation. In higher dimensions, the role of the derivative of the function is played by the Jacobian matrix (19.27), and this motivates the following definition.

Definition 19.30. A solution \( u^* \) to a system \( f(u) = 0 \) is called nonsingular if the associated Jacobian matrix is nonsingular there: \( \det f'(u^*) \neq 0 \).

Note that the Jacobian matrix is square if and only if the system has the same number of equations as unknowns, and so this is a requirement for a solution to be nonsingular. Moreover, the Inverse Function Theorem, [9, 126], from multivariable calculus implies that a nonsingular solution is necessarily isolated.

Theorem 19.31. If \( u^* \) is a nonsingular solution to the system \( f(u) = 0 \), then \( u^* \) is an isolated solution.

As with simple roots of scalar equations, nonsingular solutions of systems are the most amenable to practical computation. Non-isolated solutions, as well as isolated solutions with singular Jacobian matrices, are much more difficult to compute, and very few useful solution algorithms exist in such degenerate situations.

Now, let us turn to numerical solution techniques. The first remark is that, unlike the scalar case, proving existence of a solution to a system of equations is often a difficult problem. There is no counterpart to the Intermediate Value Theorem 19.19 for vector-valued functions; it is easy to construct examples of vector-valued functions, whose entries take on both positive and negative values, but for which there are no solutions to the system (19.45); see Exercise [ ] for one simple example. For this reason, there is no decent analog of the Bisection method for systems of equations.

On the other hand, Newton’s method can be straightforwardly adapted to compute nonsingular solutions to systems of equations, and forms the most widely used method for
this purpose. The derivation proceeds in very similar manner to the scalar case. First, we replace the system (19.45) by a fixed point system

$$u = g(u)$$  \hspace{2cm} (19.46)

having the same solutions. By direct analogy with (19.37), any (reasonable) fixed point method will take the form

$$g(u) = u - L(u)f(u),$$  \hspace{2cm} (19.47)

where $L(u)$ is an $n \times n$ matrix-valued function. Clearly, if $f(u) = 0$ then $g(u) = u$; conversely, if $g(u) = u$, then $L(u)f(u) = 0$. If we further require that the matrix $L(u)$ be nonsingular, i.e., $\det L(u) \neq 0$, then every fixed point of the iterator (19.47) will be a solution to the system (19.45) and vice versa.

According to Theorem 19.14, the speed of convergence (if any) of the iterative method

$$u^{(k+1)} = g(u^{(k)})$$  \hspace{2cm} (19.48)

is governed by the spectral radius or matrix norm of the Jacobian matrix $g'(u^*)$ at the fixed point. In particular, if

$$g'(u^*) = 0$$  \hspace{2cm} (19.49)

is the zero matrix, then the method is quadratically convergent. Computing the derivative using the matrix version of the Leibniz rule for the derivative of a matrix product, cf. Exercise II, we find

$$g'(u^*) = I - L(u^*)f'(u^*),$$  \hspace{2cm} (19.50)

where $I$ is the $n \times n$ identity matrix; see Exercise II for details. (Fortunately, all the terms that involve derivatives of the entries of $L(u)$ go away since $f(u^*) = 0$ by assumption.) Therefore, the quadratic convergence criterion (19.49) holds if and only if

$$L(u^*)f'(u^*) = I,$$  \hspace{2cm} and hence  \hspace{2cm} $L(u^*) = f'(u^*)^{-1}$  \hspace{2cm} (19.51)

should be the inverse of the Jacobian matrix of $f$ at the solution, which, fortuitously, was already assumed to be nonsingular.

As in the scalar case, we don’t know the solution $u^*$, but we can arrange that condition (19.51) holds by setting

$$L(u) = f'(u)^{-1}$$

everywhere — or at least everywhere that $f$ has a nonsingular Jacobian matrix. The resulting fixed point system

$$u = g(u) = u - f'(u)^{-1}f(u),$$  \hspace{2cm} (19.52)

leads to the quadratically convergent *Newton iteration scheme*

$$u^{(k+1)} = u^{(k)} - f'(u^{(k)})^{-1}f(u^{(k)}).$$  \hspace{2cm} (19.53)

All it requires is that we guess an initial value $u^{(0)}$ that is sufficiently close to the desired solution $u^*$. We are then guaranteed that the iterates $u^{(k)}$ converge quadratically fast to $u^*$.  

2/25/04 853 © 2004 Peter J. Olver
Theorem 19.32. Let \( u^* \) be a nonsingular solution to the system \( f(u) = 0 \). Then, provided \( u^{(0)} \) is sufficiently close to \( u^* \), the Newton iteration scheme (19.53) converges at a quadratic rate to the solution: \( u^{(k)} \rightarrow u^* \).

Example 19.33. Consider the pair of simultaneous cubic equations

\[
\begin{align*}
  f_1(u, v) &= u^3 - 3uv^2 - 1 = 0, \\
  f_2(u, v) &= 3u^2v - v^3 = 0.
\end{align*}
\]  

(19.54)

It is not difficult to prove that there are three solutions:

\[
\begin{align*}
  u_1^* &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & u_2^* &= \begin{pmatrix} -0.5 \\ 0.866025 \ldots \end{pmatrix}, & u_3^* &= \begin{pmatrix} -0.5 \\ -0.866025 \ldots \end{pmatrix}.
\end{align*}
\]

The Newton scheme relies on the Jacobian matrix

\[
f'(u) = \begin{pmatrix}
  3u^2 - 3v^2 & -6uv \\
  6uv & 3u^2 - 3v^2
\end{pmatrix}.
\]

Since \( \text{det} f'(u) = 9(u^2 + v^2) \) is non-zero except at the origin, all three solutions are nonsingular, and hence, for a sufficiently close initial value, Newton’s method will converge. We compute the inverse Jacobian matrix explicitly:

\[
f'(u)^{-1} = \frac{1}{9(u^2 + v^2)} \begin{pmatrix}
  3u^2 - 3v^2 & 6uv \\
  -6uv & 3u^2 - 3v^2
\end{pmatrix}.
\]

Hence, in this particular example, the Newton iterator (19.52) is

\[
g(u) = \begin{pmatrix} u \\ v \end{pmatrix} - \frac{1}{9(u^2 + v^2)} \begin{pmatrix}
  3u^2 - 3v^2 & 6uv \\
  -6uv & 3u^2 - 3v^2
\end{pmatrix} \begin{pmatrix} u^3 - 3uv^2 - 1 \\ 3u^2v - v^3 \end{pmatrix}.
\]

Implementing (19.53). Starting with \( \mathbf{u} \), we converge to \( \mathbf{u} \).

Remark: The alert reader may notice that in this example, we are in fact merely computing the cube roots of unity, i.e., equations (19.54) are the real and imaginary parts of the complex equation \( z^3 = 1 \) when \( z = u + iv \). A complete map of the basins of attraction converging to the three different roots has a remarkably complicated, fractal-like structure, as illustrated in Figure Newt3.

Example 19.34. A robot arm consists of two rigid rods that are joined end-to-end to a fixed point in the plane, which we take as the origin \( \mathbf{0} \). The arms are free to rotate, and the problem is to configure them so that the robots hand ends up at the prescribed position \( \mathbf{a} = (a, b)^T \). The first rod has length \( \ell \) and makes an angle \( \alpha \) with the horizontal, so its end is at position \( \mathbf{v}_1 = (\ell \cos \alpha, \ell \sin \alpha)^T \). The second rod has length \( m \) and makes an angle \( \beta \) with the horizontal, and so is represented by the vector \( \mathbf{v}_2 = (m \cos \beta, m \sin \beta)^T \). The hand at the end of the second arm is at position \( \mathbf{v}_1 + \mathbf{v}_2 \), and the problem is to find values for the angles \( \alpha, \beta \) so that \( \mathbf{v}_1 + \mathbf{v}_2 = \mathbf{a} \). To this end, we need to solve the system of equations

\[
\ell \cos \alpha + m \cos \beta = a, \quad \ell \sin \alpha + m \sin \beta = b.
\]  

(19.55)
To compute the solution, we shall apply Newton’s method. First, we compute the Jacobian matrix of the system with respect to $\alpha, \beta$, which is

$$f'(\alpha, \beta) = \begin{pmatrix} -\ell \sin \alpha & -m \sin \beta \\ \ell \cos \alpha & m \cos \beta \end{pmatrix}.$$ 

As a result, the Newton iteration equation (19.53) has the explicit form

$$\begin{pmatrix} \alpha^{(k+1)} \\ \beta^{(k+1)} \end{pmatrix} = \begin{pmatrix} \alpha^{(k)} \\ \beta^{(k)} \end{pmatrix} - \frac{1}{\ell m \sin(\beta^{(k)} - \alpha^{(k)})} \begin{pmatrix} -\ell \cos \alpha^{(k)} & m \sin \beta^{(k)} \\ -\ell \cos \alpha^{(k)} & m \sin \beta^{(k)} \end{pmatrix} \begin{pmatrix} \ell \cos \alpha^{(k)} + m \cos \beta^{(k)} - a \\ \ell \sin \alpha^{(k)} + m \sin \beta^{(k)} - b \end{pmatrix},$$

when running the iteration, one must be careful to avoid points at which $\alpha^{(k)} - \beta^{(k)} = 0$ or $\pi$, i.e., where the robot arm has straightened out.

As an example, let us assume that the rods have lengths $\ell = 2$, $m = 1$, and the desired location of the hand is at $a = (1, 1)^T$. We start with an initial guess of $\alpha^{(0)} = 0$, $\beta^{(0)} = \frac{1}{2} \pi$, so the first rod lies along the $x$-axis and the second is perpendicular. The first few Newton iterates are given in the accompanying table. The first column gives the iterate number $k$. The second and third columns indicate the angles $\alpha^{(k)}$, $\beta^{(k)}$ of the rods. The fourth and fifth give the position $(x^{(k)}, y^{(k)})^T$ of the joint or elbow, while the final two indicate the position $(z^{(k)}, w^{(k)})^T$ of the robot’s hand.

Thus, the robot has rapidly converged to one of the two possible configurations. Convergence is dependent upon the initial configuration, and the iterates do not always settle down. For instance, if $\|a\| > \ell + m$, there is no possible solution, since the arms are too short for the hand to reach to desired location; thus, no choice of initial conditions will lead to a convergent scheme and the robot arm flaps around in a chaotic manner.

Now that we have gained some experience with Newton’s method for systems of equations, some supplementary remarks are in order. As we learned back in Chapter 1, except perhaps in very low-dimensional situations, one should not invert a matrix directly, but rather use Gaussian elimination, or, in favorable situations, a linear iterative scheme, e.g., Jacobi, Gauss–Seidel or SOR, to solve a linear system. So it is better to write the Newton
equation (19.53) in unsolved, implicit form
\[ f'(u^{(k)}) v^{(k+1)} = -f(u^{(k)}), \quad u^{(k+1)} = u^{(k)} + v^{(k)}. \] (19.56)

Given the iterate \( u^{(k)} \), we first compute the Jacobian matrix \( f'(u^{(k)}) \), and then use our preferred linear systems solver to find \( v^{(k)} \). Adding \( u^{(k)} \) to the result immediately yields the updated approximation \( u^{(k+1)} \) to the solution.

Therefore, the main bottleneck in the implementation of the Newton scheme, particularly for large systems, is solving the linear system (19.56). The coefficient matrix \( f'(u^{(k)}) \) must be recomputed at each step of the iteration, and hence knowing the solution to the \( k \)th linear system does not help us solve the next one in the sequence. Having to re-implement a complete Gaussian elimination at every step will tend to slow down the algorithm, particularly in high dimensional situations involving many equations in many unknowns.

One simple dodge for speeding up the computation is to note that, once we start converging, \( u^{(k)} \) will be very close to \( u^{(k-1)} \) and so we will probably not go far wrong by using \( f'(u^{(k-1)}) \) in place of the updated Jacobian matrix \( f'(u^{(k)}) \). Since we have already solved the linear system with coefficient matrix \( f'(u^{(k-1)}) \), we know its \( LU \) factorization, and hence can use forward and back substitution to quickly solve the modified system
\[ f'(u^{(k-1)}) v^{(k+1)} = -f(u^{(k)}), \quad u^{(k+1)} = u^{(k)} + v^{(k)}. \] (19.57)

If \( u^{(k+1)} \) is still close to \( u^{(k-1)} \), we can continue to use \( f'(u^{(k-1)}) \) as the coefficient matrix when proceeding on to the next iterate \( u^{(k+2)} \). We continue until there has been a notable change in the iterates, at which stage we can revert to solving the correct, unmodified linear system (19.56) by Gaussian elimination. In this version of the algorithm, we update the coefficient matrix every few iterations, particularly if the value of the approximations has significantly changed. This device may dramatically reduce the total amount of computation required to approximate the solution to a prescribed accuracy. The down side is that this quasi-Newton scheme is only linearly convergent, and so does not home in on the root as fast as the unmodified implementation. The user needs to balance the trade-off between speed of convergence versus amount of time needed to solve the linear system at each step in the process.

19.3. Optimization.

We have already remarked on the importance of quadratic minimization principles to characterize the equilibrium solutions of a variety of linear systems. In nonlinear mathematics, optimization loses none of its centrality, and the wealth of practical applications has spawned an entire subdiscipline of applied mathematics. Physical systems naturally seek to minimize the potential energy function, and so determination of the possible equilibrium configurations requires solving a nonlinear minimization principle. Engineering design is guided by a variety of optimization constraints, such as performance, safety, cost and marketability. Non-quadratic minimization principles also arise in the fitting of data by more general schemes beyond the simple linear least squares approximation method.
discussed in Section 4.3. Additional applications arise in economics and financial mathematics — one often wishes to minimize costs or maximize profits — in manufacturing, in biological and ecological systems, in pattern recognition and signal processing, and in statistics.

The Objective Function

Throughout this section, the function \( F(u) = F(u_1, \ldots, u_n) \) to be minimized — the energy, cost, entropy, performance, etc. — will be called the objective function. As such, it depends upon one or more variables \( u = (u_1, u_2, \ldots, u_n)^T \) that belong to a prescribed subset \( \Omega \subset \mathbb{R}^n \).

**Definition 19.35.** A point \( u^* \in \Omega \) is a global minimum of the objective function on the domain \( \Omega \) if

\[
F(u^*) \leq F(u) \quad \text{for all} \quad u \in \Omega. \tag{19.58}
\]

The minimum is called strict if

\[
F(u^*) < F(u) \quad \text{for} \quad u \neq u^* \in \Omega. \tag{19.59}
\]

The point is called a local minimum if the inequality holds just for points \( u \) nearby \( u^* \), i.e., satisfying \( \| u - u^* \| < \delta \) for some \( \delta > 0 \). Thus, strict local minima are isolated.

The definition of a maximum — local or global — is the same, but with the reversed inequality: \( F(u^*) \geq F(u) \) or, in the strict case, \( F(u^*) > F(u) \). Alternatively, a maximum of \( F(u) \) is the same as a minimum of the negative \( -F(u) \). Therefore, every result that applies to minimization of a function can easily be translated into a result on maximization, which allows us to concentrate exclusively on the minimization problem without any loss of generality. We will use extremum\(^\dagger\) as a shorthand term for either a maximum or a minimum.

**Remark:** As we already noted in Section 4.1, any system of equations can be readily converted into a minimization principle. Thus, given a system (19.45), we consider the function\(^\ddagger\)

\[
F(u) = \| f(u) \|^2 = f_1(u_1, \ldots, u_n)^2 + \cdots + f_n(u_1, \ldots, u_n)^2. \tag{19.60}
\]

By the basic properties of the norm, the minimum value is \( F(u) = 0 \), and this is achieved if and only if \( f(u) = 0 \), i.e., at a solution to the system.

In contrast to the much more complicated existence question for systems of equations, there is an general theorem that guarantees the existence of minima (and, hence, maxima) for a very broad class of optimization problems.

**Theorem 19.36.** If \( F: \Omega \rightarrow \mathbb{R} \) is continuous, and \( \Omega \subset \mathbb{R}^n \) is closed and bounded, then \( F \) has at least one global minimum \( u^* \in \Omega \).

\(^\dagger\) Curiously, the term “optimum” is not used.

\(^\ddagger\) We use the standard Euclidean norm, but any other norm would work equally well here.
See [125, 126] for a proof. Although Theorem 19.36 assures us of the existence of a global minimum of any continuous function on a bounded domain, it does not guarantee uniqueness, nor does it indicate how to go about finding it. Just as with the solution of nonlinear systems of equations, it is quite rare that one can find exact formulae for the minima of non-quadratic functions. Our goal, then, is to formulate practical algorithms that can accurately compute the minima of general nonlinear functions. A naïve algorithm, but one that is often successfully applied in practical problems, [121, opt], is to select a reasonably dense set of sample points \( u^{(k)} \) in the domain and compare the values of \( f(u^{(k)}) \).

If the points are sufficiently densely distributed and the function is not too wild, this will give a good approximation to the minimum. The algorithm can be speeded up by using sophisticated methods of selecting the sample points.

As the student no doubt remembers, there are two different possible types of minima. An interior minimum occurs at an interior point of the domain of definition of the function, whereas a boundary minimum occurs on its boundary \( \partial \Omega \). Interior local minima are easier to find, and, to keep the presentation simple, we shall focus our efforts on them.

Let us review the basic procedure for optimizing scalar functions that you learned in calculus.

**Example 19.37.** Let us optimize the scalar function

\[
 f(u) = 8u^3 + 5u^2 - 6u
\]

on the domain \(-1 \leq u \leq 1\). As you learned in first year calculus, the first step to finding the minimum is to look at the critical points where the derivative vanishes:

\[
 f'(u) = 24u^2 + 10u - 6 = 0, \quad \text{and hence} \quad u = \frac{1}{3}, -\frac{3}{4}.
\]

To ascertain the local nature of the two critical points, we apply the second derivative test. Since \( f''(u) = 48u + 10, \) we have

\[
 f''\left(\frac{1}{3}\right) = 26 > 0, \quad \text{whereas} \quad f''\left(-\frac{3}{4}\right) = -26 < 0,
\]

and we conclude that \( \frac{1}{3} \) is a local minimum, while \( -\frac{3}{4} \) is a local maximum.

To find the global minimum and maximum on the interval \([-1, 1]\), we must also take into account the boundary points \( \pm 1 \). Comparing the function values at the four points,

\[
 f(-1) = 3, \quad f\left(\frac{1}{3}\right) = -\frac{31}{27} \approx -1.148, \quad f\left(-\frac{3}{4}\right) = \frac{63}{16} = 3.9375, \quad f(1) = 7,
\]

we see that \( \frac{1}{3} \) is the global minimum, whereas 1 is the global maximum. This is borne out by the graph of the function in Figure 19.5.

**The Gradient**

As the student learns in multi-variable calculus, the (interior) extrema — minima and maxima — of a smooth function \( F(u) = F(u_1, \ldots, u_n) \) are necessarily critical points, meaning places where the gradient of \( F \) vanishes. The gradient of a function is, of course, the vector whose entries are its first order partial derivatives:

\[
 \nabla F(u) = \left( \frac{\partial F}{\partial u_1}, \ldots, \frac{\partial F}{\partial u_n} \right)^T. \tag{19.61}
\]
Let us, in preparation for more general minimization problems over infinite-dimensional function spaces, reformulate the definition of the gradient in a more intrinsic manner. An important but subtle point is that the gradient operator, in fact, relies upon the introduction of an inner product on the underlying vector space. The “standard version” (19.61) is based upon on the Euclidean inner product on $\mathbb{R}^n$. Altering the inner product will change the formula for the gradient!

**Definition 19.38.** Let $V$ be an inner product space. Given a function $F: \Omega \to \mathbb{R}$ defined on an open domain $\Omega \subset V$, its gradient at a point $u \in \Omega$ is the vector $\nabla F(u) \in V$ that satisfies

$$
\langle \nabla F(u) ; v \rangle = \left. \frac{d}{dt} F(u + tv) \right|_{t=0} \quad \text{for all} \quad v \in V.
$$

(19.62)

The left hand side of (19.62) is known as the directional derivative of $F$ with respect to $v \in V$, typically denoted by $\partial F/\partial v$.

In the Euclidean case, when $F(u) = F(u_1, \ldots, u_n)$ is a function of $n$ variables, defined for $u \in \mathbb{R}^n$, we can use the chain rule to compute

$$
\frac{d}{dt} F(u + tv) = \frac{d}{dt} F(u_1 + tv_1, \ldots, u_n + tv_n) = v_1 \frac{\partial F}{\partial u_1}(u + tv) + \cdots + v_n \frac{\partial F}{\partial u_n}(u + tv).
$$

(19.63)

Setting $t = 0$, the right hand side of (19.62) reduces to

$$
\left. \frac{d}{dt} F(u + tv) \right|_{t=0} = v_1 \frac{\partial F}{\partial u_1}(u) + \cdots + v_n \frac{\partial F}{\partial u_n}(u) = v \cdot \nabla F(u) = \nabla F(u) \cdot v.
$$

Therefore, the directional derivative equals the Euclidean dot product between the usual gradient of the function (19.61) and the direction vector $v$.

A function $F(u)$ is continuously differentiable if and only if its gradient $\nabla F(u)$ is a continuously varying function of $u$. This is equivalent to the requirement that the first order partial derivatives $\partial F/\partial u_i$ are all continuous. As usual, we use $C^1(\Omega)$ to denote the vector space of all continuously differentiable scalar-valued functions defined on a domain
\[ \Omega \subset \mathbb{R}^n. \] From now on, all objective functions are assumed to be continuously differentiable on their domain of definition.

**Remark:** In this chapter, we will only deal with the standard Euclidean dot product and hence the usual gradient (19.61). However, all results can be readily translated into more general situations, e.g., weighted inner products. Details are outlined in Exercise \text{I}.

More generally, if \( \mathbf{u}(t) \) represents a parametrized curve contained within the domain of definition of \( F(\mathbf{u}) \), then the instantaneous rate of change in the scalar quantity \( F \) as we move along the curve is given by

\[
\frac{d}{dt} F(\mathbf{u}(t)) = \left( \nabla F(\mathbf{u}); \frac{d\mathbf{u}}{dt} \right),
\] (19.64)

which is the directional derivative of \( F \) with respect to the velocity or tangent vector \( \mathbf{v} = \dot{\mathbf{u}} \) to the curve. For instance, our rate of ascent or descent as we travel through the mountains is given by the dot product of our velocity vector with the gradient of the elevation function. This leads us to one important interpretation of the gradient vector.

**Theorem 19.39.** The gradient \( \nabla F \) of a scalar function \( F(\mathbf{u}) \) points in the direction of its steepest increase. The negative gradient, \(-\nabla F\), which points in the opposite direction, indicates the direction of steepest decrease.

For example, if \( F(u, v) \) represents the elevation of a mountain range at position \((u, v)\) on a map, then \( \nabla F \) tells us the direction that is steepest uphill, while \(-\nabla F\) points directly downhill — the direction water will flow. Similarly, if \( F(u, v, w) \) represents the temperature of a solid body, then \( \nabla F \) tells us the direction in which it is getting the hottest. Heat energy (like water) will flow in the opposite, coldest direction, namely that of the negative gradient vector \(-\nabla F\).

You need to be careful in how you interpret Theorem 19.39. Clearly, the faster you move along a curve, the faster the function \( F(\mathbf{u}) \) will vary, and one needs to take this into account when comparing the rates of change along different curves. The easiest way to normalize is to assume that the tangent vector \( \mathbf{a} = \dot{\mathbf{u}} \) has norm 1, so \( \| \mathbf{a} \| = 1 \) and we are passing through the point \( \mathbf{u} \) with unit speed. Once this is done, Theorem 19.39 is an immediate consequence of the Cauchy–Schwarz inequality (3.16). Indeed,

\[
\left| \frac{\partial F}{\partial \mathbf{a}} \right| = |\mathbf{a} \cdot \nabla F| \leq \| \mathbf{a} \| \| \nabla F \| = \| \nabla F \|, \quad \text{when} \quad \| \mathbf{a} \| = 1,
\]

with equality if and only if \( \mathbf{a} = c \nabla F \) points in the same direction as the gradient. Therefore, the maximum rate of change is when \( \mathbf{a} = \nabla F/\| \nabla F \| \) is the unit vector in the direction of the gradient, while the minimum is achieved when \( \mathbf{a} = -\nabla F/\| \nabla F \| \) points in the opposite direction. As a result, Theorem 19.39 tells us how to move if we wish to minimize a scalar function as rapidly as possible.

**Theorem 19.40.** A curve \( \mathbf{u}(t) \) will realize the steepest decrease in the scalar field \( F(\mathbf{u}) \) if and only if it satisfies the gradient flow equation

\[
\dot{\mathbf{u}} = -\nabla F(\mathbf{u}). \quad (19.65)
\]
**Critical Points**

Let us now prove that the gradient vanishes at any local minimum of the function. The most important thing about this proof is that it only relies on the intrinsic definition of gradient, and therefore applies to any function on any inner product space. Moreover, even though the gradient can change if we alter the underlying inner product, the condition that it vanishes at a local extremum does not.

**Definition 19.41.** A point \( \mathbf{u}^* \) is called a \textit{critical point} of the objective function \( F(\mathbf{u}) \) if

\[
\nabla F(\mathbf{u}^*) = 0.
\]

**Theorem 19.42.** If \( \mathbf{u}^* \in \Omega \) is a local (interior) minimum of \( F(\mathbf{u}) \), then \( \nabla F(\mathbf{u}^*) = 0 \), and so \( \mathbf{u}^* \) is a critical point.

**Proof:** Let \( \mathbf{v} \in \mathbb{R}^n \) be any vector. Consider the function

\[
g(t) = F(\mathbf{u}^* + t \mathbf{v}) = F(u_1^* + tv_1, \ldots, u_n^* + tv_n),
\]

where \( t \in \mathbb{R} \) is sufficiently small to ensure that \( \mathbf{u}^* + t \mathbf{v} \in \Omega \) remains inside the domain of \( F \). Thus, \( g \) measures the values of \( F \) along a straight line passing through \( \mathbf{u}^* \) in the direction\(^\dagger \) prescribed by \( \mathbf{v} \). Since \( \mathbf{u}^* \) is a local minimum,

\[
F(\mathbf{u}^*) \leq F(\mathbf{u}^* + t \mathbf{v}), \quad \text{and hence} \quad g(0) \leq g(t)
\]

for all \( t \) sufficiently close to zero. In other words, \( g(t) \), as a function of the single variable \( t \), has a local minimum at \( t = 0 \). By the basic calculus result on minima of functions of one variable, the derivative of \( g(t) \) must vanish at \( t = 0 \). Therefore, by the definition (19.62) of gradient,

\[
0 = g'(0) = \frac{d}{dt} F(\mathbf{u}^* + t \mathbf{v}) \bigg|_{t=0} = \langle \nabla F(\mathbf{u}^*); \mathbf{v} \rangle.
\]

We conclude that the gradient vector \( \nabla F(\mathbf{u}^*) \) at the critical point must be orthogonal to every vector \( \mathbf{v} \in \mathbb{R}^n \). The only vector that is orthogonal to every vector in an inner product space is the zero vector, and hence \( \nabla F(\mathbf{u}^*) = 0 \).

**Q.E.D.**

**Remark:** As we learned, the gradient vector \( \nabla F \) points in the direction of the steepest increase in the function, while its negative, \( -\nabla F(\mathbf{u}) \), points in the direction of steepest decrease. At a minimum of the function, all directions are increasing, and so there is no direction of steepest decrease. The only way that the gradient can avoid this little dilemma is for it to vanish, which provides an intuitive explanation of why minima (and maxima) must be critical points.

\(^\dagger \) If \( \mathbf{v} = 0 \), then the line degenerates to a point, but the ensuing argument remains (trivially) valid.
Thus, provided the objective function is continuously differentiable, every interior minimum, both local and global, is necessarily a critical point. The converse is not true; critical points can be maxima; they can also be saddle points or of some degenerate form. The basic analytical method for determining the (interior) minima of a given function is to first find all its critical points by solving the system of equations (19.66). Each critical point then needs to be more closely examined — as it could be either a minimum, or a maximum, or neither.

**Example 19.43.** Consider the function

$$F(u, v) = u^4 - 2u^2 + v^2,$$

which is defined and continuously differentiable on all of $\mathbb{R}^2$. Since $\nabla F = (4u^3 - 4u, 2v)^T$, its critical points are obtained by solving the system of equations

$$4u^3 - 4u = 0, \quad 2v = 0.$$ 

The solutions to the first equation are $u = 0, \pm 1$, while the second equation requires $v = 0$. Therefore, $F$ has three critical points:

$$u_1^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad u_2^* = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad u_3^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}. \quad (19.68)$$

Inspecting the graph in Figure 19.6, we suspect that the first critical point $u_1^*$ is a saddle point, whereas the other two are both global minima for the function, with the same value $F(u_2^*) = F(u_3^*) = -1$. This will be confirmed once we learn how to rigorously distinguish critical points.

---

† Numerical methods are discussed below.
If $F(u)$ is defined on a closed subdomain $\Omega \subset \mathbb{R}^n$, then its minima may also occur at boundary points $u \in \partial \Omega$, and there is no requirement that the gradient vanish at such boundary minima. The analytical determination of boundary extrema relies on the method of Lagrange multipliers, and we refer the interested reader to [9, 38]. If the domain is unbounded, one must also worry about the asymptotic behavior of the function for large $u$. In order to keep our presentation simple, we shall relegate these more involved issues to a more advanced text.

The student should also pay attention to the distinction between local minima and global minima. Both are critical points. In the absence of theoretical justification, the only practical way to determine whether or not a minimum is global is to find all the different local minima and see which one gives the smallest value. In many examples arising in applications, when $F(u)$ is often an energy function, one knows that the function is bounded from below, and hence, from general principles, that a global minimum exists, even when the domain is unbounded.

The Second Derivative Test

The status of critical point — minimum, maximum, or neither — can often be resolved by analyzing the second derivative of the objective function at the critical point. Let us first review the one variable second derivative test from first year calculus.

**Proposition 19.44.** Let $g(t) \in C^2$ be a scalar function, and suppose $t^*$ a critical point, so $g'(t^*) = 0$. If $t^*$ is a local minimum, then $g''(t^*) \geq 0$. Conversely, if $g''(t^*) > 0$, then $t^*$ is a strict local minimum. Similarly, $g''(t^*) \leq 0$ is required at a local maximum, while $g''(t^*) < 0$ implies that $t^*$ is a strict local maximum.

The proof of this result relies on the quadratic Taylor approximation

$$g(t) \approx g(t^*) + \frac{1}{2} (t - t^*)^2 g''(t^*)$$

near the critical point, (C.7), where we use the fact that $g'(t^*) = 0$ and so the linear terms in the Taylor polynomial vanish. If $g''(t^*) \neq 0$, then the quadratic Taylor polynomial has a minimum or maximum at $t^*$ according to the sign of the second derivative. In the borderline case, when $g''(t^*) = 0$, the second derivative test is inconclusive, and the point could be either maximum, minimum, saddle point, or degenerate. One must then look at the higher order terms in the Taylor expansion to resolve the issue; see Exercise [1].

In multi-variate calculus, the “second derivative” is represented by the $n \times n$ Hessian matrix

$$\nabla^2 F(u) = \begin{pmatrix}
\frac{\partial^2 F}{\partial u_1^2} & \frac{\partial^2 F}{\partial u_1 \partial u_2} & \cdots & \frac{\partial^2 F}{\partial u_1 \partial u_n} \\
\frac{\partial^2 F}{\partial u_2 \partial u_1} & \frac{\partial^2 F}{\partial u_2^2} & \cdots & \frac{\partial^2 F}{\partial u_2 \partial u_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 F}{\partial u_n \partial u_1} & \frac{\partial^2 F}{\partial u_n \partial u_2} & \cdots & \frac{\partial^2 F}{\partial u_n^2}
\end{pmatrix},$$

(19.69)
named after the early eighteenth German mathematician Ludwig Otto Hesse. The entries of the Hessian are the second order partial derivatives of the objective function. If \( F(u) \in C^2 \) has continuous second order partial derivatives, then its Hessian matrix is symmetric, \( \nabla^2 F(u) = \nabla^2 F(u)^T \), which is a restatement of the fact that its mixed partial derivatives are equal: \( \partial^2 F/\partial u_i \partial u_j = \partial^2 F/\partial u_j \partial u_i \), cf. [9, 38]. For the applicability of the second derivative test, this is an essential ingredient.

The second derivative test for a local minimum of scalar function relies on the positivity of its second derivative. For a function of several variables, the corresponding condition is that the Hessian matrix be positive definite, as in Definition 3.22. More specifically:

**Theorem 19.45.** Let \( F(u) = F(u_1, \ldots, u_n) \in C^2(\Omega) \) be a real-valued, twice continuously differentiable function defined on an open domain \( \Omega \subset \mathbb{R}^n \). If \( u^* \in \Omega \) is a (local, interior) minimum for \( F \), then it is necessarily a critical point, so \( \nabla F(u^*) = 0 \). Moreover, the Hessian matrix (19.69) must be positive semi-definite at the minimum, so \( \nabla^2 F(u^*) \geq 0 \). Conversely, if \( u^* \) is a critical point with positive definite Hessian matrix \( \nabla^2 F(u^*) > 0 \), then \( u^* \) is a strict local minimum of \( F \).

**Proof:** We return to the proof of Theorem 19.42. Given a local minimum \( u^* \), the scalar function \( g(t) = F(u^* + t v) \) in (19.67) has a local minimum at \( t = 0 \). As noted above, basic calculus tells us that its derivatives at \( t = 0 \) must satisfy

\[
g'(0) = 0, \quad g''(0) \geq 0. \tag{19.70}
\]

The first condition leads to the critical point equation \( \nabla F(u^*) = 0 \). A straightforward chain rule calculation produces the formula

\[
g''(0) = \sum_{i,j=1}^n \frac{\partial^2 F}{\partial u_i \partial u_j}(u^*) v_i v_j = v^T \nabla^2 F(u^*) v.
\]

As a result, the second condition in (19.70) requires that

\[
v^T \nabla^2 F(u^*) v \geq 0.
\]

Since this condition is required for every direction \( v \in \mathbb{R}^n \), the Hessian matrix \( \nabla^2 F(u^*) \geq 0 \) satisfies the criterion for positive semi-definiteness, proving the first part of the theorem.

Conversely, if the Hessian \( \nabla^2 F(u^*) > 0 \) is positive definite, then

\[
g''(0) = v^T \nabla^2 F(u^*) v > 0 \quad \text{for all} \quad v \neq 0,
\]

and so \( t = 0 \) is a strict local minimum for \( g(t) \). Since this occurs for every \( v \in V \), this implies \( F(u^*) < F(u) \) for all \( u \) near \( u^* \) and so \( u^* \) is a strict local minimum. \( \text{Q.E.D.} \)

---

\[\dagger\] We are ignoring some technical details that need cleaning up for a completely rigorous proof, which relies on the multivariable Taylor expansion of \( F(u) \). See Appendix C.
A maximum requires a negative semi-definite Hessian matrix. If, moreover, the Hessian at the critical point is negative definite, then the critical point is a strict local maximum. If the Hessian matrix is indefinite, then the critical point is a saddle point, and neither minimum nor maximum. In the borderline case — when the Hessian is only positive or negative semi-definite at the critical point, then the second derivative test is inconclusive. Resolving the nature of the critical point requires more detailed knowledge of the objective function, e.g., its higher order derivatives.

Example 19.46. As a first, elementary example, consider the quadratic function

$$F(u, v) = u^2 - 2uv + 3v^2.$$ 

To minimize $F$, we begin by computing its gradient $\nabla F = \begin{pmatrix} 2u - 2v \\ -2u + 6v \end{pmatrix}$. Solving the pair of equations $\nabla F = 0$, namely

$$2u - 2v = 0, \quad -2u + 6v = 0,$$

we see that the only critical point is the origin $u = v = 0$. To test whether the origin is a maximum or minimum, we further compute the Hessian matrix

$$H = \nabla^2 F(u, v) =\begin{pmatrix} F_{uu} & F_{uv} \\ F_{uv} & F_{vv} \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ -2 & 6 \end{pmatrix}.$$ 

Using the methods of Section 3.5, we easily prove that the Hessian matrix is positive definite. Therefore, by Theorem 19.45, $u^* = 0$ is a strict local minimum of $F$.

Indeed, we recognize $F(u, v)$ to be, in fact, a homogeneous positive definite quadratic form, which can be written in the form

$$F(u, v) = u^T K u,$$

where $K = \frac{1}{2} H = \begin{pmatrix} 1 & -1 \\ -1 & 3 \end{pmatrix}$ and $u = \begin{pmatrix} u \\ v \end{pmatrix}$.

Positive definiteness of the coefficient matrix $K$ implies that $F(u, v) > 0$ for all $u = (u, v)^T \neq 0$, and hence $0$ is, in fact, a global minimum.

In general, any quadratic function $Q(u) = Q(u_1, \ldots, u_n)$ can be written in the form

$$Q(u) = u^T K u - 2b^T u + c = \sum_{i,j=1}^m k_{ij} u_i u_j - 2 \sum_{i=1}^n b_i u_i + c, \quad (19.71)$$

where $K = K^T$ is a symmetric $n \times n$ matrix, $b \in \mathbb{R}^n$ is a fixed vector, and $c \in \mathbb{R}$ is a scalar. A straightforward computation produces the formula for its gradient and Hessian matrix:

$$\nabla Q(u) = 2K u - 2b, \quad \nabla^2 Q(u) = 2K. \quad (19.72)$$

As a result, the critical points of the quadratic function are the solutions to the linear system $K u = b$. If $K$ is nonsingular, there is a unique critical point $u^*$, which is a strict local minimum if and only if $K > 0$ is positive definite. In fact, Theorem 4.1 tells us that, in the positive definite case, $u^*$ is a strict global minimum for $Q(u)$. Thus, the algebraic
The approach of Chapter 4 provides additional, global information that cannot be gleaned directly from the local, multivariable calculus Theorem 19.45. But algebra is only able to handle quadratic minimization problems with ease. The analytical classification of minima and maxima of more complicated objective functions necessarily relies on the gradient and Hessian criteria of Theorem 19.45.

**Example 19.47.** The function

$$F(u, v) = u^2 + v^2 - v^3$$

has gradient

$$\nabla F(u, v) = \begin{pmatrix} 2u \\ 2v - 3v^2 \end{pmatrix}.$$ 

There are two solutions to the critical point equation $\nabla F = 0$: $u_1^* = (0, 0)^T$ and $u_2^* = (0, \frac{2}{3})^T$. The Hessian matrix of the objective function is

$$\nabla^2 F(u, v) = \begin{pmatrix} 2 & 0 \\ 0 & 2 - 6v \end{pmatrix}.$$ 

At the first critical point, the Hessian $\nabla^2 F(0, 0) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ is positive definite. Therefore, the origin is a strict local minimum. On the other hand, $\nabla^2 F\left(0, \frac{2}{3}\right) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$ is indefinite, and hence $u_2^* = \left(0, \frac{2}{3}\right)^T$ a saddle point. The function is graphed in Figure 19.7, with the critical points indicated by the small solid balls. The origin is, in fact, only a local minimum, since $F(0, 0) = 0$, whereas $F(0, v) < 0$ for all $v > 1$. Thus, there is no global minimum or maximum on $\mathbb{R}^2$.

Next, consider the function

$$F(u, v) = u^2 + v^4,$$

with gradient

$$\nabla F(u, v) = \begin{pmatrix} 2u \\ 4v^3 \end{pmatrix}.$$ 

The only critical point is the origin $u = v = 0$. The origin is a strict global minimum because $F(u, v) > 0 = F(0, 0)$ for all $(u, v) \neq (0, 0)^T$. However, its Hessian matrix

$$\nabla^2 F(u, v) = \begin{pmatrix} 2 & 0 \\ 0 & 12v^2 \end{pmatrix}.$$
is only positive semi-definite at the origin, \( \nabla^2 F(0, 0) = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \), and the second derivative test is inconclusive.

On the other hand, the origin \( u = v = 0 \) is also the only critical point for the function

\[
F(u, v) = u^2 + v^3 \quad \text{with} \quad \nabla F(u, v) = \begin{pmatrix} 2u \\ 3v^2 \end{pmatrix}.
\]

The Hessian matrix is

\[
\nabla^2 F(u, v) = \begin{pmatrix} 2 & 0 \\ 0 & 6v \end{pmatrix}, \quad \text{and so} \quad \nabla^2 F(0, 0) = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}
\]

is the same positive semi-definite matrix at the critical point. However, in this case \((0, 0)\) is not a local minimum; indeed

\[
F(0, v) < 0 = F(0, 0) \quad \text{whenever} \quad v < 0,
\]

and so there exist points arbitrarily close to the origin where \( F \) takes on smaller values. As illustrated in Figure 19.7, the origin is, in fact, a degenerate saddle point.

Finally, the function

\[
F(u, v) = u^2 - 2uv + v^2 \quad \text{has gradient} \quad \nabla F(u, v) = \begin{pmatrix} 2u - 2v \\ -2u + 2v \end{pmatrix},
\]

and so every point \( u = v \) is a critical point. The Hessian matrix

\[
\nabla^2 F(u, v) = \begin{pmatrix} F_{uu} & F_{uv} \\ F_{uv} & F_{vv} \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}
\]

is positive semi-definite everywhere. Since \( F(u, u) = 0 \), while \( F(u, v) = (u - v)^2 > 0 \) when \( u \neq v \), each of these critical points is a non-isolated local minimum, but not a strict local minimum. Thus, comparing the three preceding examples, we see that a semi-definite Hessian cannot completely distinguish critical points.

Finally, the reader should always remember that first and second derivative tests only determine the local behavior of the function near the critical point. They cannot be used to determine whether or not we are at a global minimum. This requires some additional analysis, and, often, a fair amount of ingenuity.

**Minimization of Scalar Functions**

In practical optimization, one typically bypasses the preliminary characterization of minima as critical points, and instead implements a direct iterative procedure that constructs a sequence of successively better approximations. As the computation progresses, the approximations are adjusted so that the objective function is made smaller and smaller, which, we hope, will ensure that we are converging to some form of minimum.

As always, to understand the issues involved, it is essential to consider the simplest scalar situation. Thus, we are given the problem of minimizing a scalar function \( F(u) \) on a bounded interval \( a \leq u \leq b \). The minimum value can either be at an endpoint or an interior minimum. Let us first state a result that plays a similar role to the Intermediate Value Theorem 19.19 that formed the basis of the bisection method for finding roots.
Lemma 19.48. Suppose that $F(u)$ is defined and continuous for all $a \leq u \leq b$. Suppose that we can find a point $a < c < b$ such that $F(c) < F(a)$ and $F(c) < F(b)$. Then $F(u)$ has a minimum at some point $a < u^* < b$.

The proof is an easy consequence of Theorem 19.36. Therefore, if we find three points $a < c < b$ satisfying the conditions of the lemma, we are assured of the existence of a local minimum for the function between the two endpoints. Once this is done, we can design an algorithm to home in on the minimum $u^*$. We choose another point, say $d$ between $a$ and $c$ and evaluate $F(d)$. If $F(d) < F(c)$, then $F(d) < F(a)$ also, and so the points $a < d < c$ satisfy the hypotheses of Lemma 19.48. Otherwise, if $F(d) > F(c)$ then the points $d < c < b$ satisfy the hypotheses of the lemma. In either case, a local minimum has been narrowed down to a smaller interval, either $[a, c]$ or $[d, b]$. In the unlikely event that $F(d) = F(c)$, one can try another point instead — unless the objective function is constant, one will eventually find a suitable value of $d$. Iterating the method will produce a sequence of progressively smaller and smaller intervals in which the minimum is trapped, and, just like the bisection method, the endpoints of the intervals get closer and closer to $u^*$.

The one question is how to choose the point $d$. We described the algorithm when it was selected to lie between $a$ and $c$, but one could equally well try a point between $c$ and $b$. To speed up the algorithm, it makes sense to place $d$ in the larger of the two subintervals $[a, c]$ and $[c, b]$. One could try placing $d$ in the midpoint of the interval, but a more inspired choice is to place it at position $\frac{a + c}{2}$. The result is the Golden Section Method, and is outlined in the accompanying program. At each stage, the length of the interval has been reduced by a factor of $\frac{1}{\sqrt{5}} \approx 0.61803$. Thus, the convergence rate is linear, and a bit slower than the bisection algorithm.

Another strategy is to use an interpolating polynomial through the three points on the graph of $F(u)$ and use the minimum value of that polynomial as the next approximation to the minimum. According to Exercise 19.48, the minimizing value is at

$$d = \frac{ms - nt}{s - t},$$

where

$$s = \frac{F(c) - F(a)}{c - a}, \quad t = \frac{F(b) - F(c)}{b - c}, \quad m = \frac{a + c}{2}, \quad n = \frac{c + b}{2}.$$ 

As long as $a < c < b$ satisfy the hypothesis of Lemma 19.48, we are assured that the quadratic interpolant has a minimum (and not a maximum!), and that the minimum remains between the endpoints of the interval. If the length of the interval is small, the minimum value should be a good approximation to the minimizer $u^*$ of $F(u)$ itself. Once $d$ is determined, the algorithm proceeds as before. In this case, convergence is not quite guaranteed, or, in unfavorable situations, could be much slower than in the preceding method. One can even try using the method when the function values do not satisfy the hypothesis of Lemma 19.48, although now the new point $d$ will not necessarily lie between $a$ and $b$. Worse, the quadratic interpolant may have a maximum at $d$, and one ends up going in the wrong direction, which can even happen in the minimizing case due to the
discrepancy between it and the objective function $F(u)$. Thus, this case must be handled with more caution, and convergence of the scheme is much more fraught with danger.

A final idea is to focus not on the objective function $F(u)$ but rather its derivative $f(u) = F'(u)$. The critical points of $F$ are the roots of $f(u) = 0$, and so one can use one of the solution methods, e.g., bisection or Newton’s method, to find the critical points. Of course, one must then take care that the critical point $u^*$ is indeed a minimum, as it could equally well be a maximum of the original objective function. (It will probably not be a saddle point, as these do not correspond to simple roots of $f(u)$.) But this can be checked by looking at the sign of $F''(u^*) = f'(u^*)$ at the root; indeed, if we use Newton’s method we will be computing the derivative at each stage of the algorithm, and can stop looking if the derivative is of the wrong sign.

*Gradient Descent*

Now, let us turn our attention to multi-dimensional optimization problems. We are seeking to minimize a (smooth) scalar objective function $F(u) = F(u_1, \ldots, u_n)$. According to Theorem 19.39, at any given point $u$ in the domain of definition of $F$, the negative gradient vector $-\nabla F(u)$, if nonzero, points in the direction of the steepest decrease in $F$. Thus, to minimize $F$, an evident strategy is to “walk downhill”, and, to be efficient, walk downhill as fast as possible, namely in the direction $-\nabla F(u)$. After walking in this direction for a little while, we recompute the gradient, and this tells us the new direction to head downhill. With luck, we will eventually end up at the bottom of the valley, i.e., at a (local) minimum value of the objective function.

This simple idea forms the basis of the *gradient descent* method for minimizing the objective function $F(u)$. In a numerical implementation, we start the iterative procedure with an initial guess $u^{(0)}$, and let $u^{(k)}$ denote the $k$th approximation to the minimum $u^*$. To compute the next approximation, we move away from $u^{(k)}$ in the direction of the negative gradient, and hence

$$u^{(k+1)} = u^{(k)} - t_k \nabla F(u^{(k)}) \quad (19.73)$$

for some positive scalar $t_k > 0$ that indicates how far we move in the negative gradient direction. We are free to adjust $t_k$ so as to optimize our descent path, and this is the key to the success of the method.

If $\nabla F(u^{(k)}) \neq 0$, then, at least when $t_k > 0$ is sufficiently small,

$$F(u^{(k+1)}) < F(u^{(k)}), \quad (19.74)$$

and so $u^{(k+1)}$ is, presumably, a better approximation to the desired minimum. Clearly, we cannot choose $t_k$ too large or we run the risk of overshooting the minimum and reversing the inequality (19.74). Think of walking downhill in the Swiss Alps. If you walk too far in a straight line, which is what happens as $t_k$ increases, then you might very well miss the valley and end up higher than you began — not a good strategy for descending to the bottom! On the other hand, if we choose $t_k$ too small, taking very tiny steps, then the method will converge to the minimum much too slowly.
How should we choose an optimal value for the factor $t_k$? Keep in mind that the goal is to minimize $F(u)$. Thus, a good strategy would be to set $t_k$ equal to the value of $t > 0$ that minimizes the scalar objective function

$$g(t) = F(u^{(k)} - t \nabla F(u^{(k)}))$$

(19.75)

obtained by restricting $F(u)$ to the ray emanating from $u^{(k)}$ that lies in the negative gradient direction. Physically, this corresponds to setting off in a straight line in the direction of steepest decrease in our altitude, and continuing on in this direction until we cannot go down any further. Barring luck, we will not have reached the actual bottom of the valley, but must then readjust our direction and continue on down the hill in a series of straight line paths, each connecting $u^{(k)}$ to $u^{(k+1)}$.

In practice, one can rarely compute the minimizing value $t^*$ of $g(t)$ exactly. Instead, we use one of the scalar minimization algorithms presented in the previous subsection. Note that we only need to look for a minimum among positive values of $t > 0$, since our choice of the negative gradient direction assures us that, at least for $t$ sufficiently small and positive, $g(t) < g(0)$.

**Example 19.49.**

*Conjugate Gradients*

The one complication with the basic gradient descent method is that it may take a long time to reach the minimum. This is a danger if the scalar factors $t_k$ are small, and we end up taking very tiny steps in each round of the iteration. This occurs if we are looking for a minimum in a long narrow valley, as illustrated in Figure valley. The initial step takes us into the valley, but then we spend a long time meandering back and forth along the valley floor before we come close to the true minimum.

One method to avoid such difficulties, and speed up the convergence rate of the scheme, is to use the method of *conjugate directions*, modeled on the quadratic minimization procedure discussed in in Section 16.2.