Chapter 1 Solving Equations

§1.1 Introductin (Propaedeutics)

We now consider the most basic of tasks, the *root-finding problem,* solving equations numerically. While most equations are born with both a right-hand side and a left-hand side, one traditionally moves all terms to the left, leaving

$$f(x) = 0 \qquad (1.1.1)$$

whose solution or solutions are desired. When there is only one independent variable, the problem is *one-dimensional*, namely to find the root or roots of a function.

We give some important basic facts from calculus as follows.

Theorem 1.1.1(Intermediate Value Theorem)

Let f be a continuous function on the interval [a, b](that is $f \in C[a. b]$). Then f realizes every value between f(a) and f(b). More precisely, if y is a number between f(a) and f(b), then there exists a number ξ with $a \le \xi \le b$ such that $f(\xi) = y$.

Theorem 1.1.2(Continuous Limits)

Let f be a continuous function in a neighborhood of x_0, and assume $\lim_{n\to\infty} x_n = x_0$. Then

$$\lim_{n\to\infty} f(x_n) = f(\lim_{n\to\infty} x_n) = f(x_0)$$

In other words, limits may be brought inside continuous functions.

Theorem 1.1.3 (Mean Value Theorem)

Let f be a continuously differentiable function on the interval [a, b] (that is $f \in C^1[a, b]$). Then there exists a number ξ between a and b such that $f'(\xi) = (f(b) - f(a)) / (b - a)$.

Theorem 1.1.4 (Rolle's Theorem)

Let $f \in C^1[a. b]$) and assume that f(a) = f(b). Then there exists a number ξ between a and b such that $f'(\xi) = 0$.

Theorem 1.1.5 (Taylor's Theorem with Remainder)

Let x and x_0 be real numbers, and let f be k+1 times continuously differentiable on the interval between x and x_0 (that is $f \in C^{k+1}[a, b]$). Then there exists a number ξ between x and x_0 such that

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \cdots$$

$$+ \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \frac{f^{(k+1)}(\xi)}{(k+1)!}(x - x_0)^{k+1}$$

in which $\frac{f^{(k+1)}(\xi)}{(k+1)!}(x-x_0)^{k+1}$ is called the remainder.

§1.2 The Bisection Method

This process involves finding a *root*, or solution, of an equation of the form (1.1.1), for a given function f. A root of this equation is also called a *zero* of the function f.

The first technique, based on the Intermediate Value Theorem, is called the ***Bisection***, or ***Binary-search***, method.

Theorem 1.2.1

Let $f \in C[a. b]$, satisfying $f(a)f(b) < 0$. Then f has a root between a and b, that is, there exists a number ξ satisfying $a<\xi<b$ and $f(\xi) = 0$.

Proof.

Suppose $f \in C[a, b]$, with f(a) and f(b) of opposite sign. By the Intermediate Value Theorem (Theorem 1.1.1), there exists a number ξ in (a, b) with $f(\xi) = 0$.

Although the procedure will work when there is more than one root in the interval (a, b), we assume for simplicity that the root in this interval is unique. The method calls for a repeated halving of subintervals of [a, b] and, at each step, locating the half containing ξ.

To begin, set $a_1=a$ and $b_1=b$, and let ξ_1 be the midpoint of [a, b]; that is,

$$\xi_1 = a_1 + \frac{b_1-a_1}{2} = \frac{a_1+b_1}{2}$$

If $f(\xi_1) = 0$, then $\xi=\xi_1$, and we are done. If $f(\xi_1)\neq 0$, then $f(\xi_1)$ has the same sign as either $f(a_1)$ or $f(b_1)$. When $f(\xi_1)$ and $f(a_1)$ have the same sign, $\xi \in (\xi_1,b_1)$, and we set $a_2=\xi_1$ and $b_2 = b_1$. When $f(\xi_1)$ and $f(a_1)$ have opposite signs, $\xi \in (a_1, \xi_1)$ and we set $a_2=a_1$ and $b_2=\xi_1$. We then reapply the process to the interval $[a_2, b_2]$. This produces the method described in Algorithm 1.2.1

Algorithm 1.2.1 : Bisection

To find a solution to f(x) = 0 given the continuous function f on the interval [a, b], where f(a) and f(b) have opposite signs:

INPUT endpoints a, b; tolerance TOL; maximum number of iterations N_0,

OUTPUT approximate solution p or message of failure.

Step 1 set i = 1;
FA = f(a);

Step 2 While $i < N_0$ do Steps 3-6

Step 3 Set c = (a+b) / 2
FC = f(c);

Step 4 If FC = 0 or (b-a) / 2 < TOL then
OUTPUT (c); //Procedure completed successfully.
Stop;

Step 5 Set i = i + 1;
Step 6 If FA * FC > 0 then set a = c; FA = FC;
else set b = c;
Step 7 OUTPUT('Method failed after N_0 iterations, N_0 = ',N_0);
STOP;

Other stopping procedure can be applied in Step 4 of Algorithm 1.2.1 or in any of the iterative techniques in this chapter. For example, we can select a tolerance $\varepsilon>0$ and generate $c_1, c_2, \ldots, c_N$ until one of the following conditions is met:

$$|c_N - c_{N-1}| < \varepsilon, \quad (1.2.1)$$

$$|c_N - c_{N-1}| / |c_N| < \varepsilon, \ c_N \neq 0, \text{ or} \quad (1.2.2)$$

$$|f(c_N)| < \varepsilon. \quad (1.2.3)$$

Unfortunately, difficulties can arise using any of these stopping criteria. For example, there are sequences $\{c_n\}$ with the property that differences $c_n - c_{n-1}$ converges to close to zero while c_n differs significantly from c. Without additional knowledge about f or c. Inequality (1.2.2) is the best stopping criterion to apply because it comes closest to testing relative error.

When using a computer to generate approximations, it is good practice to set an upper bound on the number of iterations. This will eliminate the possibility of entering an infinite loop, a situation that can arise when the sequence diverges (and also when the program is incorrectly coded). This was done in Step 2 of Algorithm 1.2.1 where the bound N_0 was set and the procedure terminated if $i > N_0$.

Note that to start the Bisection Algorithm, an interval [a, b] must be found with f(a)f(b) < 0. At each step the length of the interval known to contain a zero of f is reduced by a factor of 2; hence it is advantageous to choose the initial interval [a, b] as small as possible. For example, if $f(x) = 2x^3 - x^2 + x - 1$, we have both

f(-4)f(4) < 0 and f(0)f(1) < 0

so the Bisection Algorithm could be used on either of the intervals [-4,4] or [0, 1]. Starting the Bisection Algorithm on [0, 1] instead of [-4, 4] will reduce by 3 the number of iterations required to achieve a specified accuracy.

The following example illustrates the Bisection Algorithm. The iteration in this example is terminated when the relative error is less than 0.0001; that is, when

$$|c - c_n| / |c| < 10^{-4}.$$

Example 1.2.1 The equation $f(x) = x^3 + 4x^2 - 10 = 0$ has a root in [1, 2] since f(1) = -5 and f(2) = 14. The Bisection Algorithm gives the values in Table 1.2.1

Table 1.2.1

n	a_n	b_n	c_n	$f(c_n)$
1	1	2	1.5	2.375

2	1	1.5	1.25	-1.79687
3	1.25	1.5	1.375	0.16211
4	1.25	1.375	1.3125	-0.84839
5	1.3125	1.375	1.34375	-0.35098
6	1.34375	1.375	1.359375	-0.09641
7	1.359375	1.375	1.3671875	0.03236
8	1.359375	1.3671875	1.36328125	0.03215
9	1.36328125	1.3671875	1.365234375	0.000072
10	1.36328125	1.365234375	1.364257813	-0.01605
11	1.364257813	1.365234375	1.364746094	-0.00799
12	1.364746094	1.365234375	1.364900235	-0.00396
13	1.364990235	1.365234375	1.365112305	-0.00194

After 13 iterations, $c_{13} = 1.365112305$ approximates the root c with an error $|c - c_{13}| < |b_{14} - a_{14}| = |1.365234375 - 1.365112305| = 0.000122070$.
Since $|a_{14}| < |c|$,

$$|c - c_{13}| / |c| < |b_{14} - a_{14}| / |a_{14}| \leq 9.0 \times 10^{-5},$$

so the approximation is correct to at least significant digits. The correct value of c, to nine decimal places, is $c = 1.365230013$. Note that c_9 is closer to c than is the final approximation c_{13}. You might suspect this is true since $|f(c_9)| < |f(c_{13})|$, but we cannot be sure of this unless the true answer is known.

The Bisection method, though conceptually clear, has significant drawbacks. It is slow to converge (that is, N may become quite large before $|c - c_N|$ is sufficiently small), and a good intermediate approximation can be inadvertently discarded. However, the method has the important property that it always converges to a solution, and for that reason it is often used as a starter for the more efficient methods we will present later in this chapter.

Theorem 1.2.2 Suppose that $f \in C[a, b]$, and $f(a)f(b) < 0$. The Bisection method generates a sequence approximating a zero c of f with

$$|c_n - c| \leq (b - a) / 2^n \text{ , when } n \geq 1.$$

Proof. For each $n \geq 1$, we have

$$b_n - a_n = (b - a) / 2^{n-1} \quad \text{and } c \in (a_n, b_n).$$

Since $c_n = (a_n + b_n)/2$ for all $n \geq 1$, it follows that

$$|c_n - c| \leq (b_n - a_n)/2 = (b - a)/ 2^n .$$

Since

$$|c_n - c| \leq (b - a)/ 2^n$$

The sequence converges to c with rate of convergence $O(1/2^n)$; that is,

$$c_n = c + O(1/2^n).$$

It is important to realize that Theorem 1.2.1 gives only a bound for approximation error and that this bound may be quite conservative. For example, this bound applied to the problem in Example 1.2.1

ensures only that

$$|c - c_9| \le (2 - 1)/2^9 \approx 2 \times 10^{-3},$$

but the actual error is much smaller:

$$|c - c_9| \le |1.365230013 - 1.365234375| \approx 2 \times 10^{-6}.$$

§1.3 Fixed-Point Iteration (FPI)

§1.3.1 Fixed points of a function

Definition 1.3.1

The real number x^* is a **fixed point** of the function φ if $\varphi(x^*) = x^*$.

In this section we consider the problem of finding solutions to fixed-point problems and the connection between the fixed-point problems and the root-finding problems we wish to solve.

Root-finding problems and fixed-point problems are equivalent classes in the following sense:

Given a root-finding problem $f(x) = 0$, we can define functions φ with a fixed point at x^* in a number of ways, for example, as $\varphi(x) = x - f(x)$ or as $\varphi(x) = x + 3f(x)$. Conversely, if the function φ has a fixed point at x^*, then the function defined by $f(x) = x - \varphi(x)$ has a zero at x^*.

Although the problems we wish to solve are in the root-finding form, the fixed-point form is easier to analyze, and certain fixed-point choices lead to very powerful root-finding techniques.

We first need to become comfortable with this new type of problem and to decide when a function has a fixed point and how the fixed points can be approximated to within a specified accuracy.

Example 1.3.1

The function $\varphi(x) = x^2 - 2$, for $-2 \le x \le 3$, has fixed point at $x=-1$ and $x=2$ since

$$\varphi(-1) = (-1)^2 - 2 = -1 \quad \text{and} \quad \varphi(2) = 2^2 - 2 = 2.$$

This can be seen in Figure 1.3.1.

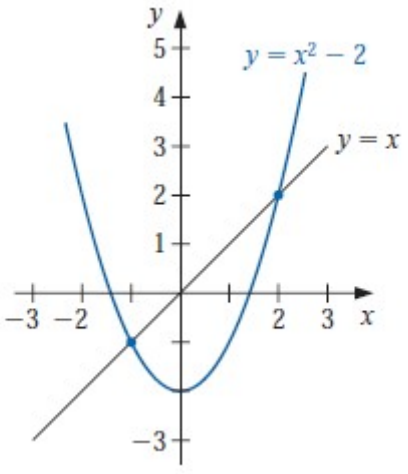

Figure 1.3.1

The following theorem gives sufficient conditions for the existence and uniqueness of a fixed point.

Theorem 1.3.1

a. If $\varphi \in C[a, b]$, and $\varphi(x) \in [a, b]$, for all x in [a, b], then φ has a fixed point in [a, b].
b. If, in addition, $\varphi'(x)$ exists on (a, b) and a positive constant $k < 1$ exists with
$$|\varphi'(x)| \le k, \text{ for all } x \in (a, b).$$
then the fixed point in [a, b] is unique. (See Figure 1.3.2.)

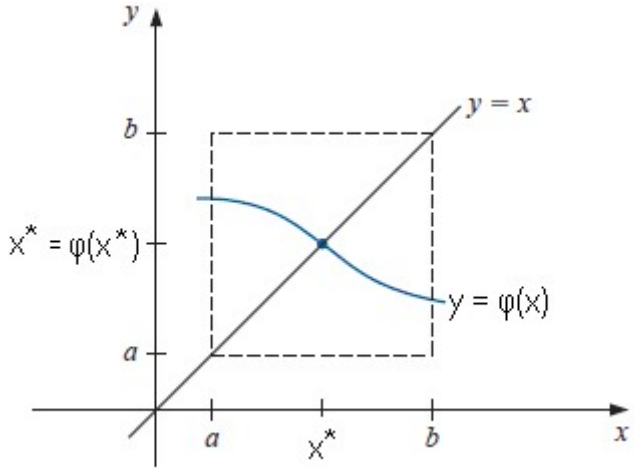

Figure 1.3.2

Proof

a. If $\varphi(a) = a$ or $\varphi(b) = b$, then φ has a fixed point at an endpoint. If not, then $\varphi(a) > a$ and $\varphi(b) < b$. The function $h(x) = \varphi(x) - x \in C[a, b]$, with
$$h(a) = \varphi(a) - a > 0 \text{ and } h(b) = \varphi(b) - b < 0.$$
The Intermediate value Theorem (Throrem 1.1.1) implies that there exists $x^* \in (a, b)$ for which $h(x^*) = 0$. This number x^* is a fixed point for φ since
$$0 = h(x^*) = \varphi(x^*) - x^* \quad \text{implies that } \varphi(x^*) = x^*.$$
b. Suppose, in addition, that $|\varphi'(x)| \le k < 1$ and that x_1 and x_2 are both fixed points in [a, b]. If $x_1 \neq x_2$, then the Mean value Theorem (Theorem 1.1.3) implies that a number ξ exists between x_1 and x_2, and hence in [a, b], with
$$\frac{\varphi(x_1) - \varphi(x_2)}{x_1 - x_2} = g'(\xi).$$
Thus,
$|x_1 - x_2| = |\varphi(x_1) - \varphi(x_2)| = |\varphi'(\xi)||x_1 - x_2| \le k|x_1 - x_2| < |x_1 - x_2|$,
which is a contradiction. This contradiction must come from the only supposition, $x_1 \neq x_2$. Hence, $x_1 = x_2$ and the fixed point

in [a, b] is unique.

Example 1.3.2

a. Let $\varphi(x) = (x^2 - 1) / 3$ on [-1, 1]. The Extreme Value Theorem implies that the absolute minimum of φ occurs at x = 0 and φ(0) = -1/3. Similarly, the absolute maximum of φ occurs at x = ±1 and has the value φ(±1) = 0. Moreover, φ is continuous and

$$|\varphi'(x)| = |2x/3| \le 2/3, \text{ for all } x \in (-1,1).$$

So φ satisfies all the hypotheses of Theorem 1.3.1 and has a unique fixed point in [-1,1].

In this example, the unique fixed point x^* in the interval [-1, 1] can be determined algebraically. If

$$x^* = \varphi(x^*) = (x^{*2} - 1) / 3, \text{ then } x^{*2} - 3x^* - 1 = 0,$$

which, by the quadratic formula, implies that

$$x^* = \frac{1}{2}(3 - \sqrt{13}).$$

Note that φ also has a unique fixed point $x^* = \frac{1}{2}(3 + \sqrt{13})$ for the interval [3, 4]. However, φ(4) = 5 and φ'(4) = 8/3 > 1, so φ does not satisfy the hypotheses of Theorem 1.3.1 on [3,4]. Hence, the hypotheses of Theorem 1.3.1 are sufficient to guarantee a unique fixed point but are not necessary. (See Figure 1.3.3.)

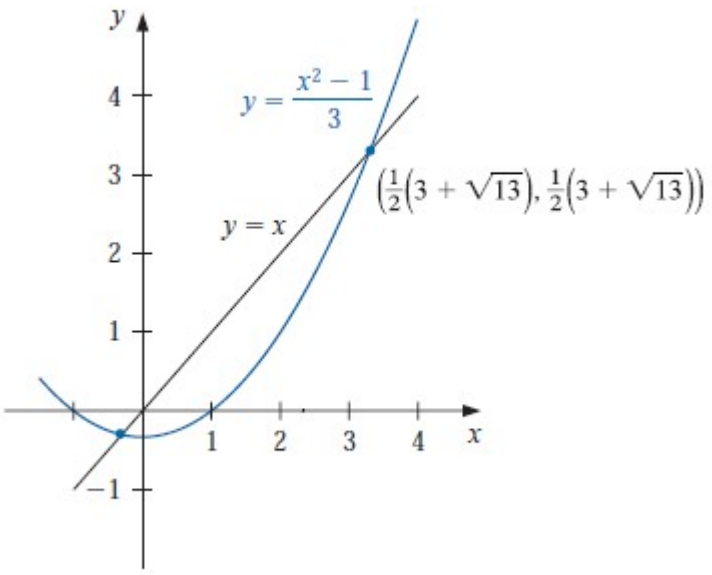

Figure 1.3.3

b. Let $\varphi(x) = 3^{-x}$. Since $\varphi'(x) = -3^{-x}\ln 3 < 0$ on [0, 1], the function φ is decreasing on [0,1]. So

$$\varphi(1) = 1/3 \le \varphi(x) \le 1 = \varphi(0), \quad \text{for } 0 \le x \le 1.$$

Thus, for $x \in [0, 1]$, we have $\varphi(x) \in [0, 1]$, and φ has a fixed point in [0, 1]. Since

$$\varphi'(0) = -\ln 3 = -1.098612289,$$

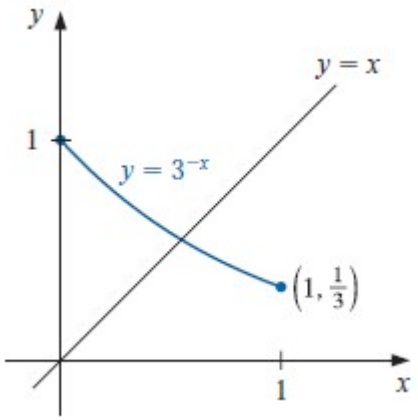

Figure 1.3.4

|φ'(x)| not less than 1 on (0, 1), and Theorem 1.3.1 cannot be used to determine uniqueness. However, φ is always decreasing, and it is clear from Figure 1.3.4 that the fixed point must be unique.

To approximate the fixed point of a function φ, we choose an initial approximation x_0 and generate the sequence $\{x_n\}$ by letting $x_n = \varphi(x_{n-1})$, for each $n \geq 1$. If the sequence converges to x^* and g is continuous, then

$$x^* = \lim_{n\to\infty} x_n = \lim_{n\to\infty} \varphi(x_{n-1}) = \varphi(\lim_{n\to\infty} x_{n-1}) = g(x^*),$$

and a solution to x = φ(x) is obtained. This technique is called **fixed-point iteration**, or **functional iteration**. That is:

x_0 is initial value

$x_i = \varphi(x_{i-1})$ $\quad i \geq 1$

Algorithm 1.3.1 fixed-point iteration

To find a solution to $x^* = \varphi(x^*)$ given an initial approximation x_0:

INPUT initial approximation x_0; tolerance TOL; maximum number of iterations N_0.

OUTPUT approximate solution x^* or message of failure.

Step 1 Set i = 1;

Step 2 While $i \leq N_0$ do Step 3 – 6.

Step 3 Set $x^* = \varphi(x_0)$.

Step 4 if $|x^* - x_0| < TOL$ then

OUTPUT(x^*); //The procedure was successful.

Stop.

Step 5 Set i = i + 1

Step 6 Set $x_0 = x^*$.

Step 7 OUTPUT('The method failed after N_0 iterations, N_0=', N_0);

Stop;

The following example illustrates fixed-point iteration.

Example 1.3.3

The equation $x^3 + x - 1 = 0$ has unique root (approximation root:

0.68233) in [0,1]. There are many ways to change the equation to the fixed-point form x = φ(x) using simple algebraic manipulation. For example,

a. can be rewritten as $x = 1 - x^3 = \varphi(x)$,

b. and $x = \sqrt[3]{1-x} = \varphi(x)$.

c. add $2x^3$ to both sides of the equation. So, $x = \frac{1+2x^3}{1+3x^2} = \varphi(x)$

With $x_0 = 0.5$, the results of the fixed-point iteration for all three choices of φ are Table 1.3.1.

Table 1.3.1

$\varphi(x)= 1 - x^3$		$\varphi(x)= \sqrt[3]{1-x}$				$\varphi(x)=\frac{1+2x^3}{1+3x^2}$	
i	x_i	i	x_i	i	x_i	i	x_i
0	0.5	0	0.5	13	0.68454401	0	0.5
1	0.875	1	0.79370053	14	0.68073737	1	0.71428571
2	0.33007813	2	0.59088011	15	0.68346460	2	0.68317972
3	0.96403747	3	0.74236393	16	0.68151292	3	0.68232842
4	0.10405419	4	0.63631020	17	0.68291073	4	0.68232780
5	0.99887338	5	0.71380081	18	0.68191019	5	0.68232780
6	0.00337606	6	0.65900615	19	0.68262667	6	0.68232780
7	0.99999996	7	0.69863261	20	0.68211376	7	0.68232780
8	0.00000012	8	0.67044850	21	0.68248102		
9	1	9	0.69072912	22	0.68221809		
10	0	10	0.67625892	23	0.68240635		
11	1	11	0.68664554	24	0.68227157		
12	0	12	0.67922234	25	0.68236807		

a. the method is divergent. After the 8^{th} , 0 and 1 will be alternately appearance.

b. The method is convergent. After the 25^{th}, $x_i \to 0.682368$.

c. The method is also convergent. After the 4^{th}, $x_i \to 0.6823278$.

Even though the various functions in Example 1.3.3 are fixed-point problems for the same root-finding problem, they differ vastly as techniques for approximating the solution to the root-finding problem. Their purpose is to illustrate the true question that needs to be answered:

How can we find a fixed-point problem that produces a sequence that reliably and rapidly converges to a solution to a given root-finding problem?

The following theorem gives us some clues concerning the paths we should pursue and, perhaps more importantly, some we should reject.

Theorem 1.3.2 (Fixed-Point Theorem)

Let $\varphi \in C[a,b]$, $\varphi(x) \in [a,b]$, for all x in [a,b]. Suppose, in addition, that φ' exists on (a,b) and that a constant $0<k<1$ exists with

$$|\varphi'(x)| \le k, \text{ for all } x \in (a,b).$$

Then, for any number x_0 in [a, b], the sequence defined by

$$x_n = \varphi(x_{n-1}), \quad n \ge 1,$$

Converges to the unique fixed point x^* in [a,b].

Proof

Theorem 1.3.1 implies that a unique fixed point exists in [a, b]. Since φ maps [a, b] into itself, the sequence $\{x_n\}$ is defined for all $n \ge 0$, and $x_n \in [a,b]$ for all n. Using the fact that $|\varphi'(x)| \le k$ and the Mean Value Theorem, we have, for each n,

$$|x_n - x^*| = |\varphi(x_{n-1}) - \varphi(x^*)| = |\varphi'(\xi_n)||x_{n-1} - x^*| \le k|x_{n-1} - x^*|,$$

where $\xi_n \in (a, b)$. Applying this inequality inductively gives

$$|x_n - x^*| \le k|x_{n-1} - x^*| \le k^2|x_{n-2} - x^*| \le \ldots \le k^n|x_0 - x^*|$$

Since $0 < k < 1$, we have $\lim_{n\to\infty} k^n = 0$ and

$$\lim_{n\to\infty} |x_n - x^*| \le \lim_{n\to\infty} k^n |x_0 - x^*| = 0$$

Hence, $\{x_n\}$ converges to x^*.

§1.3.2 Geometry of Fixed-Point Iteration

FPI procedure is detailed and illustrated in Figure 1.3.5.

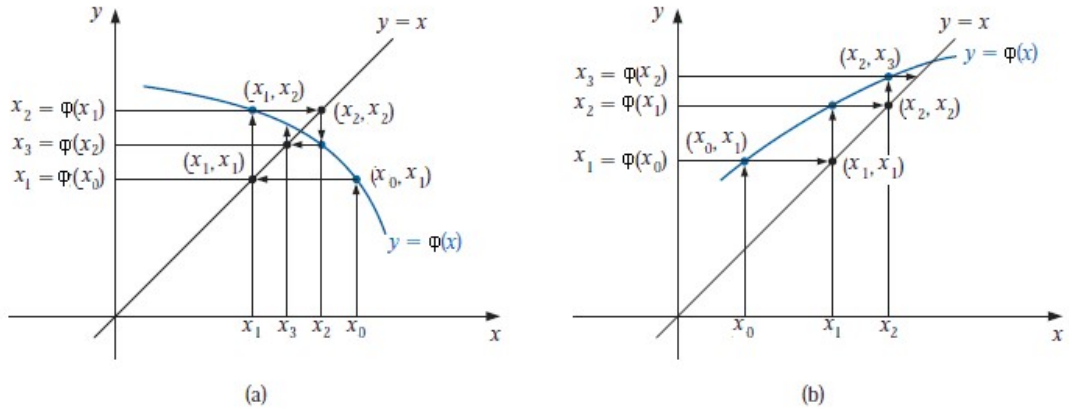

Figure 1.3.5

The fixed point x^* is represented by the point where the graph $y = \varphi(x)$ and $y = x$ intersect. Each step of Fixed-point iteration can be sketched by drawing line segments (1) vertically to the function and then (2) horizontally to the diagonal line $y = x$. The vertical and horizontal arrows in Figure 1.3.5 follow the steps made by Fixed Point Iteration. The vertical arrow moving from the x-value to the function φ represents $x_i = \varphi(x_i)$. The horizontal arrow represents turning the output $\varphi(x_i)$ on the y-axis and transforming it into the same number x_{i+1} on the x-axis, ready to be input into φ in the next step. This is

done by drawing the horizontal line segment from the output height $\varphi(x_i)$ across to the diagonal line y = x. The geometric illustration of a Fixed-Point Iteration as called a **cobweb diagram**.

In the previous section, we found three different ways to rewrite the equation $x^3 + x - 1 = 0$ as fixed-point problem, with varying results. To find out why the Fixed Point Iteration method converges in some situations and not in others, it is helpful to look at the geometry of the method.

In Figure 1.3.6, the path starts at $x_0 = 0.5$, and as we saw earlier, the result of Fixed Point Iteration for this φ(x) is not successful—the iterates eventually tend toward alternating between 0 and 1, neither of which are fixed points.

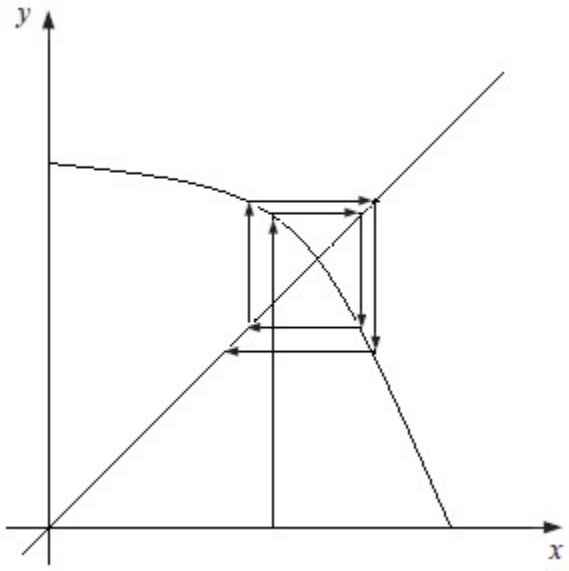

Figure 1.3.6 divergence.

§1.3.3 Linear convergence of Fixed-Point Iteration

The convergence properties of FPI can be easily explained by a careful look at the algorithm in the simplest possible situation. Figure 1.3.7 shows FPI for two linear functions $\varphi_1(x) = -\frac{3}{2}x + \frac{5}{2}$ and $\varphi_2(x) = -\frac{1}{2}x + \frac{3}{2}$. In each case, the fixed point is x = 1, but $|\varphi'_1(x)| = |-\frac{3}{2}| > 1$ while $|\varphi'_2(x)| = |-\frac{1}{2}| < 1$. Following the vertical and horizontal arrows that describe FPI, we see the reason for the difference. Because the slope of φ_1 at the fixed point is greater than one, the vertical segments, the ones that represent the change from x_n to x_{n+1}, are increasing in length as FPI proceeds. As a result, the represent the iteration "spirals out" from the fixed point $x^* = 1$, even if the initial guess x_0 was quite near. For φ_2, the situation is reversed: The slope of φ_2 is less than one, the vertical segments decrease in length, and FPI "spirals in" toward

the solution. Thus, $|\varphi'(x^*)|$ makes the crucial difference between divergence and convergence.

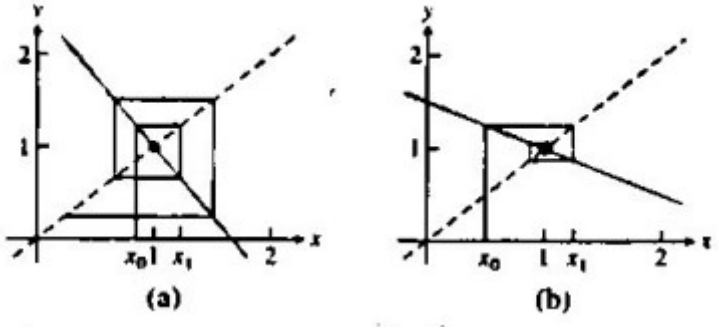

Figure 1.3.7 Cobweb diagram for linear function.

(a) if the linear function has slope greater than one in absolute value, nearby guessed move farther from the fixed point as FPI progresses, leading to failure of the method.

(b) For slope less than one in absolute value, the reverse happens, and the fixed point is found.

That's the geometric view. In terms of equations, it helps to write $\varphi_1(x)$ and $\varphi_2(x)$ in terms of $x - x^*$, where $x^* = 1$ is the fixed point:

$$\varphi_1(x) = -\frac{3}{2}(x-1)+1$$

$$\varphi_1(x) - 1 = -\frac{3}{2}(x-1)$$

$$x_{i+1} - 1 = -\frac{3}{2}(x_i - 1) \qquad (1.3.1)$$

If we view $e_i = |x^* - x_i|$ as the error at step i (meaning the distance from the best guess at step n to the fixed point), we see from (1.3.1) that $e_{i+1} = 3e_i/2$, implying that errors increase at each step by a factor of approximately 3/2. This is divergence.

Repeating the preceding algebra for φ_2, we have

$$\varphi_2(x) = -\frac{1}{2}(x-1)+1$$

$$\varphi_2(x) - 1 = -\frac{1}{2}(x-1)$$

$$x_{i+1} - 1 = -\frac{1}{2}(x_i - 1) \qquad (1.3.2)$$

The result is $e_{i+1} = e_i / 2$, implying that the error, the distance to the fixed point, is multiplied by 1/2 on each step. The error decreases to zero as the number of steps increases. This is convergence of a particular type.

Definition 1.3.2

Let e_i denote the error at step i of an iterative method. If

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = s < 1$$

the method is said to obey **linear convergence** with rate s.

Fixed Point Iteration for φ_2 is linearly convergent to the root $x^* = 1$ with rate s = 1/2. Although the previous discussion was simplified because φ_1 and φ_2 are linear, the same reasoning applies to a general continuously differentiable function φ(x) with fixed point $\varphi(x^*) = x^*$, as shown in the next theorem.

Theorem 1.3.3

Assume that φ is continuously differentiable, that $\varphi(x^*) = x^*$. and that $s = |\varphi'(x^*)| < 1$. Then FPI converges linearly with rate s to the fixed point x^* for initial guesses sufficiently close to x^*.

Proof.

Let x_i denote the iterate at step i. According to the Mean Value Theorem, there exists a number c_i between x_i and x^* such that

$$x_{i+1} - x^* = \varphi'(c_i)(x_i - x^*) \qquad (1.3.3)$$

where we have substituted $x_{i+1} = \varphi(x_i)$ and $x^* = \varphi(x^*)$. Defining $e_i = |x_i - x^*|$, (1.3.3) can be written as

$$e_{i+1} = |\varphi'(c_i)|e_i \qquad (1.3.4)$$

If $s = |\varphi'(x^*)|$ is less than one, then by the continuity of φ', there is a small neighborhood around x^* for which $|\varphi'(x)| < (s+1)/2$, slightly larger than s, but still less than one. If x_i happens to lie in this neighborhood, then c_i does, too (it is trapped between x_i and x^*), and so

$$e_{i+1} \le \frac{s+1}{2} e_i$$

Thus, the error decreases by a factor of (s+1)/2 or better on this and every future step. That means $\lim_{i\to\infty} x_i = x^*$, and taking the limit of (1.3.4) yields

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = \lim_{i\to\infty} |\varphi'(c_i)| = |\varphi'(x^*)| = s$$

Example 1.3.4

a. For $\varphi(x) = 1 - x^3$, we have φ(0) = 1, φ(1) = 0. Hence φ(x) does map [0, 1] into itself. Moreover, $\varphi'(x) = -3x^2$, so $|\varphi'(x)|>1$ when 1>x>0.58. Although Theorem 1.3.2 does not guarantee that the method must fail for this choice of g, there is no reason to expect convergence.

b. For $\varphi(x) = \sqrt[3]{1-x}$, we have $\varphi'(x) = -\frac{1}{3}(1-x)^{-\frac{2}{3}}$, $|\varphi'(x)|<1$ when

0<x<0.8, so, Theorem 1.3 confirms the convergence.

c. For $\varphi(x)=\frac{1+2x^3}{1+3x^2}$, $\varphi'(x) = \frac{6x(x^3+x-1)}{(1+3x^2)^2}$, so, $|\varphi'(x)| < 1$ when $x\in(0,1)$. The iteration is convergence.

Definition 1.3.3

An iterative method is called **locally convergent** to x* if the method converges to p for initial guesses sufficiently close to x*.

Example 1.3.5

1. Explain why the FPI $\varphi(x) = \cos x$ converges.
2. Find the fixed points of $\varphi(x) = 2.8x - x^2$.
3. calculate $\sqrt{2}$ by using FPI.

§1.3.4 Stopping criteria

Unlike the case of bisection, the number of steps required for FPI to converge within a given tolerance is rarely predictable beforehand. In the absence of an error formula for the Bisection Method, a decision must be made about terminating the algorithm, called a **stopping criterion**.

For a set tolerance, TOL, we may ask for an absolute error stopping criterion

$$|x_{i+1} - x_i| < TOL \qquad (1.3.5)$$

Or, in case the solution is not too near zero, the relative error stopping criterion

$$|x_{i+1} - x_i| / |x_{i+1}| < TOL \qquad (1.3.6)$$

§1.4 Newton's method

Newton's (or the Newton-Raphson) **method** is one of the most powerful and well-known numerical methods for solving a root-finding problem. There are many ways of introducing Newton's method. If we only want an algorithm, we can consider the technique graphically, as is often done in calculus. The geometric picture of Newton's Method is shown in Figure 1.4.1. To find a root of f(x)=0, a starting guess x_0 is given, and the tangent line to the function f at x_0 is drawn. The tangent line will approximately follow the function down to the x-axis toward the root. The intersection point of the line with the x-axis is an approximate root, but probably not exact if f curves. Therefore, this step is iterated.

From the geometric picture, we can develop an algebraic formula for Newton's Method. The tangent line at x_0 has slope given by the derivative $f'(x_0)$. One point on the tangent line is $(x_0,f(x_0))$. The point-slope formula for the equation of a line is $y - f(x_0) = f'(x_0)(x- x_0)$, so that looking for the intersection point of the tangent line with the

x-axis is the same as substituting y = 0 in the line:

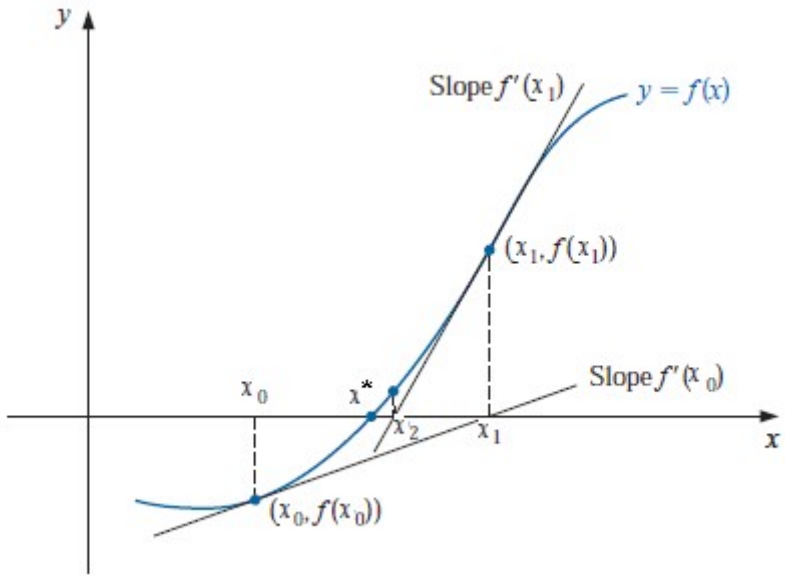

Figure 1.4.1 Two Steps of Newton's Method.

$f'(x_0)(x - x_0) = 0 - f(x_0)$

$x - x_0 = -f(x_0) / f'(x_0)$

$x = x_0 - f(x_0) / f'(x_0)$.

Solving for x gives an approximation for the root, which we call x_1. Next, the entire process is repeated, beginning with x_1, to produce x_2, and so on, yielding the following iterative formula.

Newton's Method.

x_0 = initial guess

$x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1})$, for $n \geq 1$.

Algorithm 1.4.1 Newton's

To find a solution to f(x) = 0 given an initial approximation x_0:

INPUT: initial approximation x_0; tolerance TOL; maximum number of iterations N_0.

OUTPUT: approximation solution x* or message of failure.

Step 1 Set i = 1

Step 2 while $i \leq N_0$ do Steps 3 - 6

Step 3: Set $x^* = x_0 - f(x_0) / f'(x_0)$.

Step 4: If $|x^* - x_0| < TOL$ then

OUTPUT (x*); // The procedure was successful.)

Stop.

Step 5: Set i = i + 1;

Setp 6: Set $x_0 = x^*$;

Ste[7: OUTPUT ("The method failed after N_0 iteration, N_0=",N_0);

STOP; //The procedure was unsuccessful.)

The stopping technique inequalities given with the Bisection method are applicable to Newton's method. That is, select a tolerance $\varepsilon > 0$, and construct $x_1, \ldots, x_N$ until

$$| x_N - x_{N-1}| < \varepsilon \qquad (1.4.1)$$

$$\frac{| x_N - x_{N-1} |}{| x_N |} < \varepsilon, \qquad x_N \neq 0, \qquad (1.4.2)$$

Or

$$|f(x_N)| < \varepsilon. \qquad (1.4.3)$$

A form Inequality (1.4.1) is used in Step 4 of Algorithm 1.4.1. Note that inequality (1.4.3) may not give much information about the actual error $|x_N - x^*|$.

Newton's method is a functional iteration technique of the form $x_n = \varphi(x_{n-1})$, for which

$$x_n = \varphi(x_{n-1}) = x_{n-1} - f(x_{n-1}) / f'(x_{n-1}), \quad \text{for } n \geq 1. \quad (1.4.4)$$

In fact, this is the functional iteration technique that was used to give the rapid convergence we saw in part (c) of of example 1.3.3 .

It is clear from equation (1.4.4) that Newton's method cannot be continued if $f'(x_{n-1}) = 0$ for some n. In fact, we will see that the method is most effective when f' is bounded away from zero near x*.

Example 1.4.1.

Suppose we would like to approximate a fixed point of $\varphi(x) = \cos(x)$. The graph in Figure 1.4.2 implies that a single fixed point x* lies in $[0, \pi/2]$.

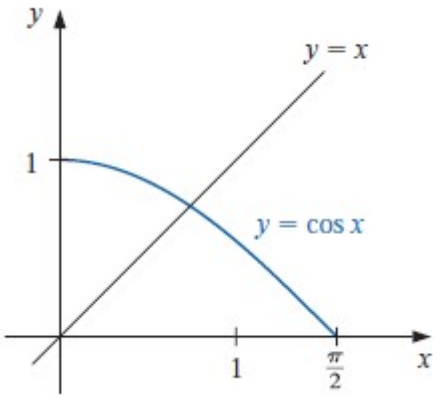

Figure 1.4.2

Let $x_0 = \pi/4$, the result of fixed-point iteration after 7^{th} iteration is 0.7361282565.

To approach this problem differently, define $f(x) = \cos(x) - x$ and apply Newton's method. Since $f'(x) = -\sin(x) - 1$, the sequence is generated by

$$x_n = x_{n-1} - (\cos(x_{n-1}) - x_{n-1})/(-\sin(x_{n-1})-1), \text{ for } n \geq 1.$$

With $x_0 = \pi/4$, the result (0.7390851332) of the approximation is

generated after 3th iteration. An excellent approximation is obtained with n = 3.

The Taylor series derivation of Newton's method at the beginning of the section points out the importance of an accurate initial approximation. The crucial assumption is that the term involving $(x^* - x')^2$ is, by comparison with $|x^* - x'|$, so small that it can be deleted. This will clearly be false unless x' is a good approximation to x*. If x_0 is not sufficiently close to the actual root, there is little reason to suspect that Newton's method will converge to the root. However, in some instances, even poor initial approximations will produce convergence.

The following convergence theorem for Newton's method illustrates the theoretical importance of the choice of x_0.

Theorem 1.4.1

Let $f \in C^2[a, b]$. If $x^* \in [a, b]$ is such that $f(x^*) = 0$ and $f'(x^*) \neq 0$, then there exists a $\delta > 0$ such that Newton's method generates a sequence $\{x_n\}$ converging to x* for any initial approximation $x_0 \in [x^* - \delta, x^* + \delta]$.

Proof

The proof is based on analyzing Newton's method as the functional iteration scheme $x_n = \varphi(x_{n-1})$, for $n \geq 1$, with

$$\varphi(x) = x - f(x) / f'(x).$$

Let k be in (0, 1). We first find an interval $[x^* - \delta, x^* + \delta]$ that φ maps into itself and for which $|\varphi'(x)| \leq k$, for all $x \in (x^* - \delta, x^* + \delta)$.

Since f' is continuous and $f'(x^*) \neq 0$, therefore, there exists a $\delta_1 > 0$, such that $f'(x) \neq 0$ for $x \in [x^* - \delta_1, x^* + \delta_1] \subseteq [a, b]$. Thus, φ is defined and continuous on $[x^* - \delta_1, x^* + \delta_1]$. Also,

$$\varphi'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2} \quad ,$$

for $x \in [x^* - \delta_1, x^* + \delta_1]$, and, since $f \in C^2[a, b]$, we have $\varphi \in C^1[x^* - \delta_1, x^* + \delta_1]$.

By assumption, $f(x^*) = 0$, so

$$\varphi'(x^*) = \frac{f(x^*)f''(x^*)}{[f'(x^*)]^2} = 0 \quad .$$

Since φ' is continuous and $0 < k < 1$, there exists a δ, with $0 < \delta < \delta_1$, and

$$|\varphi'(x)| \leq k, \text{ for } x \in [x^* - \delta, x^* + \delta].$$

It remains to show that φ maps $[x^* - \delta, x^* + \delta]$ into $[x^* - \delta, x^* + \delta]$. If $x \in [x^* - \delta, x^* + \delta]$, the Mean Value Theorem implies that for some number ξ between x and x*, $|\varphi(x) - \varphi(x^*)| = |\varphi'(\xi)||x - x^*|$. So

$$|\varphi(x) - x^*| = |\varphi(x) - \varphi(x^*)| = |\varphi'(\xi)||x - x^*| \leq k\,|x - x^*| < |x - x^*|.$$

Since $x \in [x^* - \delta, x^* + \delta]$, it follows that $|x - x^*| < \delta$. Hence, φ maps $[x^* - \delta, x^* + \delta]$ into $[x^* - \delta, x^* + \delta]$.

All the hypotheses of the Fixed Point Theorem are now satisfied, so the sequence $\{x_n\}$, defined by

$$x_n = \varphi(x_{n-1}) = x_{n-1} - f(x_{n-1})/f'(x_{n-1}), \text{ for } n \geq 1,$$

Converges to x^* for any $x_0 \in [x^* - \delta, x^* + \delta]$.

Theorem 1.4.1 states that, under reasonable assumptions, Newton's method converges provided a sufficiently accurate initial approximation is chosen. It also implies that the constant k that bounds the derivative of φ, and, consequently, indicates the speed of convergence of the method, decreases to 0 as the procedure continues. This result is important for the theorem of Newton's method, but it is seldom applied in practice since it does not tell us how to determine δ. In a practical application, an initial approximation is selected, and successive approximations are generated by Newton's method. These will generally either converge quickly to the root, or it will be clear that convergence is unlikely.

Newton's method is an extremely powerful technique, but it has a major weakness: the need to know the value of the derivative of f at each approximation. Frequently, f'(x) is far more difficult and needs more arithmetic operations to calculate than f(x).

§1.5 The Order of Convergence for Iterative Methods

In this section we investigate the order of convergence of functional iteration schemes and, as a means of obtaining rapid convergence, rediscover Newton's method. We also consider ways of accelerating the convergence of Newton's method in special circumstances. First, however, we need a procedure for measuring how rapidly a sequence converges.

Definition 1.5.1 Supose $\{x_n\}$ is a sequence that converges to x^*, with $x_n \neq x^*$ for all n. If positive constants λ and α exist with

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^{\alpha}} = \lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^{\alpha}} = \lambda \ ,$$

then $\{x_n\}$ **converges to x^* of order α, with asymptotic error constant λ.**

An iterative technique of the form $x_n = \varphi(x_{n-1})$ is said to be of order α if the sequence $\{x_n\}$ converges to the solution $x^* = \varphi(x^*)$ of order α.

In general, a sequence with a high order of convergence

converges more rapidly than a sequence with a lower order. The asymptotic constant affects the speed of convergence but is not as important as the order. Two cases of order are given special attention.

(i) If $\alpha = 1$, the sequence is **linearly convergent.**

(ii) If $\alpha = 2$, the sequence is **quadratically convergent.**

The next example compares a linearly convergent sequence to one that is quadratically convergent. It shows why we try to find methods that produce higher-order convergent sequence.

Example 1.5.1

Suppose that $\{x_n\}$ is linearly convergence to 0 with

$$\lim_{n\to\infty} \frac{|x_{n+1}|}{|x_n|} = 0.5$$

and that $\{x'_n\}$ is quadratically convergent to 0 with the same asymptotic error constant,

$$\lim_{n\to\infty} \frac{|x'_{n+1}|}{|x'_n|^2} = 0.5 \quad .$$

For simplicity, suppose that

$$\frac{|x_{n+1}|}{|x_n|} \approx 0.5 \qquad \text{and} \qquad \frac{|x'_{n+1}|}{|x'_n|^2} \approx 0.5 \quad .$$

For the linearly convergent scheme, this means that

$|x_n - 0| = |x_n| \approx 0.5|x_{n-1}| \approx (0.5)^2|x_{n-2}| \approx \ldots \approx (0.5)^n|x_0|$,

Whereas the quadratically convergent procedure has

$|x'_n - 0| = |x'_n| \approx 0.5|x'_{n-1}|^2 \approx (0.5)[0.5|x_{n-2}|^2]^2 = (0.5)^3|x_{n-2}|^4$

$\approx (0.5)^3[(0.5)|x_{n-3}|^2]^4 = (0.5)^7|x_{n-3}|^8 \approx \ldots \approx (0.5)^{2^n-1}|x'_0|^{2^n}$.

Quadratically convergent sequences generally converge much more quickly than those that converge only linearly, but many techniques that generate convergent sequences do so only linearly.

The easiest way to construct a fixed point problem associated with a root-finding problem $f(x) = 0$ is to subtract a multiple of $f(x)$ from x. So let us consider

$x_n = \varphi(x_{n-1})$, for $n \geq 1$,

for φ in the form

$\varphi(x) = x - \varphi(x) f(x)$,

where φ is a differentiable function that will be chosen later.

For the iterative procedure derived from φ to be quadratically convergent, we need to have $\varphi'(x^*)=0$ when $f(x^*) = 0$. Since

$\varphi'(x) = 1 - \varphi'(x) f(x) - f'(x) \varphi(x)$,

we have

$\varphi'(x^*) = 1 - \varphi'(x^*)f(x^*) - f'(x^*)\,\varphi(x^*) = 1 - f'(x^*)\,\varphi(x^*)$,
and $\varphi'(x^*) = 0$ if and only if $\varphi(x^*) = 1/f'(x^*)$.

If we let $\varphi(x) = 1 / f'(x)$, then we will ensure that $\varphi(x^*) = 1 / f'(x^*)$ and produce the quadratically convergent procedure

$x_n = \varphi(x_{n-1}) = x_{n-1} - f(x_{n-1}) / f'(x_{n-1})$.

This, of course, is simply Newton's method.

In the preceding discussion, the restriction was made that $f'(x^*) \neq 0$, when x^* is the solution to $f(x) = 0$. From the definition of Newton's method, it is clear that difficulties might occur if $f'(x_n)$ goes to zero simultaneously with $f(x_n)$. In particular, Newton's method will generally give problems if $f'(x^*) = 0$ when $f(x^*) = 0$. To examine these difficulties in more detail, we make the following definition.

Definition 1.5.2

A solution x^* of $f(x) = 0$ is a **zero of multiplicity** m of f if for $x \neq x^*$, we can write $f(x) = (x - x^*)^m q(x)$, where $\lim_{x \to x^*} q(x) \neq 0$.

In essence, q(x) represents that portion of f(x) that does not contribute to the zero of f. The following result gives a means to easily identify simple zeros of a function, those that have multiplicity one.

Theorem 1.5.1

$f \in C^1[a, b]$ has a simple zero at x^* in (a, b) if and only if $f(x^*) = 0$, but $f'(x^*) \neq 0$.

Proof.

If f has a simple zero at x^*, then $f(x^*) = 0$ and $f(x) = (x - x^*)q(x)$, where $\lim_{x \to x^*} q(x) \neq 0$. Since $f \in C^1[a, b]$,

$$f'(x^*) = \lim_{x \to x^*} f'(x) = \lim_{x \to x^*} [q(x) + (x - x^*)q'(x)] = \lim_{x \to x^*} q(x) \neq 0 \quad .$$

Conversely, if $f(x^*) = 0$, but $f'(x^*) \neq 0$, expand f in a zeroth Taylor polynomial about x^*. Then

$f(x) = f(x^*) + f'(\xi(x))(x-x^*) = (x - x^*)f'(\xi(x))$,

where $\xi(x)$ is between x and x^*. Since $f \in C^1[a, b]$,

$$\lim_{x \to x^*} f'(\xi(x)) = f'(\lim_{x \to x^*} \xi(x)) = f'(x^*) \neq 0 \quad .$$

Letting $q = f' \circ \xi$ gives $f(x) = (x - x^*)q(x)$, where $\lim_{x \to x^*} q(x) \neq 0$. Thus, f has a simple zero at x^*.

The following generalization of Theorem 1.5.1 is considered.

Theorem 1.5.2

The function $f \in C^m[a, b]$ has a zero of multiplicity m at x^* in (a, b) if and only if $0 = f(x^*) = f'(x^*) = f''(x^*) = \ldots = f^{(m-1)}(x^*)$, but $f^{(m)}(x^*) \neq 0$.

The result in Theorem 1.5.2 implies that an interval about x^* exists where Newton's method converges quadratically to x^* for any

initial approximation x_0, provided that x* is a simple zero. The following example shows that quadratic convergence may not occur if the zero is not simple.

Example 1.5.2

Consider $f(x) = e^x - x - 1$. Since $f(0) = e^0 - 0 - 1 = 0$ and $f'(0) = e^0 - 1 = 0$, but $f''(0) = e^0 = 1$, f has a zero of multiplicity two at $x^* = 0$. In fact, f(x) can b expressed in the form

$f(x) = (x - 0)^2(e^x - x - 1) / x^2$

where,

$$\lim_{x\to 0} \frac{e^x - x - 1}{x^2} = \lim_{x\to 0} \frac{e^x - 1}{2x} = \lim_{x\to 0} \frac{e^x}{2} = \frac{1}{2} \neq 0$$

The terms generated by Newton's method applied to f with $x_0 = 1$ are shown in Table 1.5.1. The sequence is clearly converging to 0, but not quadratically. The graph of f is shown in Figure 1.5.1.

Table 1.5.1

n	x_n	n	x_n
0	1.0	9	0.002775
1	0.58198	10	0.0013881
2	0.31906	11	0.00069411
3	0.16800	12	0.00034703
4	0.08635	13	0.00017416
5	0.04380	14	0.000088041
6	0.02206	15	0.000042610
7	0.01107	16	0.0000019142
8	0.005545		

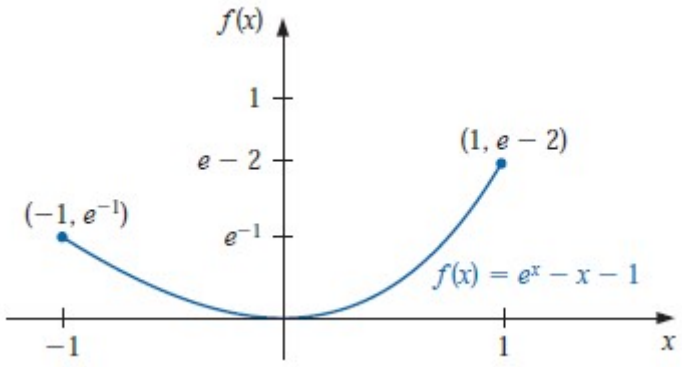

Figure 1.5.1

One method of handing the problem of multiple roots is to define

$$\mu(x) = \frac{f(x)}{f'(x)} \quad .$$

If x* is a zero of f of multiplicity m and $f(x) = (x - x^*)^m q(x)$, then

$$\mu(x) = \frac{(x-x^*)^m q(x)}{m(x-x^*)^{m-1} q(x)+(x-x^*)^m q'(x)} = (x - x^*)\frac{q(x)}{mq(x)+(x-x^*)q'(x)}$$

also has a zero at p. However, q(x*) $\neq$ 0, so

$$\frac{q(x^*)}{mq(x^*)+(x^*-x^*)q'(x^*)} = \frac{1}{m} \neq 0 \quad ,$$

And x* is a simple zero of μ. Newton's method can then be applied to μ to give

$$\varphi(x) = x - \frac{\mu(x)}{\mu'(x)} = x - \frac{f(x)/f'(x)}{\{[f'(x)]^2 - [f(x)][f''(x)]\}/[f'(x)]^2}$$

or

$$\varphi(x) = x - \frac{f(x)/f'(x)}{[f'(x)]^2 - f(x)f''(x)} \qquad (1.5.1)$$

If φ has the required continuity conditions, functional iteration applied to φ will be quadratically convergent regardless of the multiplicity of the zero of f. Theoretically, the only drawback to this method is the additional calculation of f''(x) and the more laborious procedure of calculating the iterates. In practice, however, multiple roots can cause serious roundoff problems since the denominator of (1.5.1) consists of the difference of two numbers that are both close to 0.

Exercise

1. Let f(x) = 3(x+1)(x-0.5)(x-1). Use the Bisection method on the following intervals to find x_3.
   a. [-2,1.5] b. [-1.25,2.5]
2. Use algebraic manipulation to show that each of the following functions has fixed point at x* precisely when f(x*) = 0, where f(x) = $x^3 + 4x^2 - 10$.
   a. $\varphi_1(x) = x - x^3 - 4x^2 + 10$
   b. $\varphi_2(x) = \left(\frac{10}{x} - 4x\right)^{1/2}$
   c. $\varphi_3(x) = (10 - x^3)^{1/2}/2$
   d. $\varphi_4(x) = (10/(4+x))^{1/2}$
   e. $\varphi_5(x) = x - (x^3 + 4x^2 - 10) / (3x^2 + 8x)$
   Perform four iterations if possible, on each of the function φ, Let x_0=1.5.
3. Let f(x) = $x^2 - 6$ and $x_0 = 1$. Use Newton's method to find x_2
4. Find the Fixed-Point Iteration produced by applying Newton's Method to f(x) = $x^3 - A$.

Chapter 2 Solution of Linear Equations

§2.1 Introduction

Linear systems of equations are associated with many problems in engineering and science, as well as with applications of mathematics to the social science and the quantitative study of business and economic problems.

In this chapter, direct technique and iterative technique are considered to solve the linear system

$$\begin{aligned} E_1 &: a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=b_1 \\ E_2 &: a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n=b_2 \\ &\ldots \\ E_n &: a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n=b_n \end{aligned} \qquad (2.1.1)$$

for $x_1,\ldots,x_n$, given the constants a_{ij}, for each i, j = 1,2,…,n, and b_i, for each i = 1,2,…,n. Direct technique is methods that give an answer in a fixed number of steps, subject only to roundoff errors. And the Iterative technique is also methods that give an approximative answer in limited steps. In the presentation we shall also introduce some elementary notions from the subject of linear algebra.

If we write the coefficients a_{ij} as a matrix, and the right-hand sides b_i as a column vector,

$$A=\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \qquad b=\begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix} \qquad (2.1.2)$$

then equations (2.1.1) can be written in matrix form as

$$A\cdot\mathbf{x} = \mathbf{b} \qquad (2.1.3)$$

Here, we use a dot to denote matrix multiplication, or the multiplication of a matrix and a vector, or the dot product of two vectors. Certainly, sometimes we can also omit the dot.

§2.2 Gauss Elimination with Backsubstitution

We use three operations to simplify the linear systems given in (2.1.1):

1. Equation E_i and E_j can be transposed in order. This operation is denoted $(E_i)\leftrightarrow(E_j)$.

2. Equation E_j can be multiplied by any constant λ and added to equation E_i with the resulting equation used in place of E_i. This operation is denoted $(E_i+\lambda E_j)\rightarrow(E_i)$.

3. Equation E_i can be multiplied by any nonzero constant λ with the resulting equation used in place of E_i. This operation is denoted $(\lambda E_i)\rightarrow(E_i)$.

By a sequence of these operations, a linear system can be transformed to a more easily solved linear system that has same solutions. The sequence of operations is illustrated in the next example.

Example 2.2.1 The four equations:

$$E_1 : \ x_1 + x_2 \quad + \quad 3x_4=4$$

$$E_2 : 2x_1 + x_2 \ - x_3+ x_4=1$$

$$E_3 : 3x_1 - x_2 \ - x_3+2x_4=-3$$

$$E_4 : -x_1 + 2x_2+3x_3 - x_4=4$$

will be solved for x_1, x_2, x_3, and x_4. We construct augmented matrix as follows,

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right]$$

and then, use equation E_1 to eliminate the unknown x_1 from E_2, E_3, and E_4 by performing $(E_2-2E_1) \rightarrow(E_2)$, $(E_3-3E_1) \rightarrow(E_3)$,$(E_4+E_1) \rightarrow(E_4)$.

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right] \xrightarrow[(E_4+E_1)\rightarrow(E_4)]{\substack{(E_2-2E_1)\rightarrow(E_2) \\ (E_3-3E_1)\rightarrow(E_3)}} \left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 0 & -1 & -1 & -5 & -7 \\ 0 & -4 & -1 & -7 & -15 \\ 0 & 3 & 3 & 2 & 8 \end{array}\right]$$

$$\xrightarrow[(E_4+3E_2)\rightarrow(E_4)]{(E_3-4E_2)\rightarrow(E_3)} \left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 0 & -1 & -1 & -5 & -7 \\ 0 & 0 & 3 & 13 & 13 \\ 0 & 0 & 0 & -13 & -13 \end{array}\right] \qquad (2.2.1)$$

The system of equations (2.2.1) is now in **triangular** (or **reduced**) **form** and can be solved for the unknowns by a **backward-substitution process**. Therefore, the solution to (2.2.1) is $x_1=-1$, $x_2=2$, $x_3=0$, and $x_4=1$.

Example 2.2.2 Consider augmented matrix :

$$\begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 2 & -2 & 3 & -3 & | & -20 \\ 1 & 1 & 1 & 0 & | & -2 \\ 1 & -1 & 4 & 3 & | & 4 \end{bmatrix},$$

performing the operations $(E_2\text{-}2E_1)\rightarrow(E_2)$, $(E_3\text{-}E_1)\rightarrow(E_3)$, and $(E_4\text{-}E_1)\rightarrow(E_4)$, gives

$$\begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 2 & -2 & 3 & -3 & | & -20 \\ 1 & 1 & 1 & 0 & | & -2 \\ 1 & -1 & 4 & 3 & | & 4 \end{bmatrix} \begin{matrix} (E_2\text{-}2E_1)\rightarrow(E_2) \\ (E_3\text{-}E_1)\rightarrow(E_3) \\ (E_4\text{-}E_1)\rightarrow(E_4) \end{matrix} \longrightarrow \begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 0 & 0 & -1 & -1 & | & -4 \\ 0 & 2 & -1 & 1 & | & 6 \\ 0 & 0 & 2 & 4 & | & 12 \end{bmatrix}.$$

Since a_{22}, called the **pivot element**, is zero, the procedure cannot continue in its present form. But the operation $(E_i)\leftrightarrow(E_j)$ is permitted, so a search for the first nonzero element is made.

$$\begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 0 & 0 & -1 & -1 & | & -4 \\ 0 & 2 & -1 & 1 & | & 6 \\ 0 & 0 & 2 & 4 & | & 12 \end{bmatrix} \xrightarrow{(E2)\leftrightarrow(E3)} \begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 0 & 2 & -1 & 1 & | & 6 \\ 0 & 0 & -1 & -1 & | & -4 \\ 0 & 0 & 2 & 4 & | & 12 \end{bmatrix}$$

Since x_2 is already eliminated from E_3 and E_4, and the computations continue with the operation $(E_4 + 2E_3) \rightarrow (E_4)$, giving

$$\begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 0 & 2 & -1 & 1 & | & 6 \\ 0 & 0 & -1 & -1 & | & -4 \\ 0 & 0 & 2 & 4 & | & 12 \end{bmatrix} \xrightarrow{(E4+2E3)\rightarrow(E4)} \begin{bmatrix} 1 & -1 & 2 & -1 & | & -8 \\ 0 & 2 & -1 & 1 & | & 6 \\ 0 & 0 & -1 & -1 & | & -4 \\ 0 & 0 & 0 & 2 & | & 4 \end{bmatrix}$$

Finally, the backward substitution is applied: $x_4=2$, $x_3=2$, $x_2 = 3$, $x_1=-7$.

Example 2.2.2 illustrates what is done if $a_{kk} = 0$ for some $k = 1,2,\ldots,n-1$, The kth column of $A^{(k-1)}$ from the kth row to the nth row is searched for the first nonzero entry. If $a_{pk} \neq 0$ for some p, with $k+1 \leq p \leq n$, then the operation $(E_k) \leftrightarrow (E_p)$ is performed to obtain $A^{(k-1)}$. The procedure can then be continued to form $A^{(k)}$, and so on. If $a_{pk}=0$ for

each p, it can be shown that the linear system does not have a unique solution, and again the procedure stops.

Algorithm 2.2.1 Gaussian Elimination with Backward Substitution

To solve the n×n linear system (2.1.1),

INPUT:

number of unknowns and equations n; augmented matrix $A = (a_{ij})$, where $1 \le i \le n$, $1 \le j \le n+1$.

OUTPUT:

solution $x_1, x_2, \ldots, x_n$ or message that the linear system has no unique solution.

```
//Elimination process
Step 1: for (i=1; i<n; i++) {
    Step 2:  //Let p be the smallest integer with i≤p≤n and a_pi≠0
        for (p=i; p<=n; p++)
            if (a[p,i] ≠ 0) then
                break;
        //If no integer p can be found
        //then OUTPUT ('no unique solution exists"); stop.
        if (p > n) then
            OUTPUT ("no unique solution exists"); stop.
    Step 3: if p ≠ i then perform (E_p) ↔(E_i)
    Step 4: for (j=i+1; j<=n; j++) {
            Step 5: m_ji = a[j,i]/a[i,i];
            Step 6: perform (E_j-m_jiE_i) →(E_j)
                for (k=j+1;k<=n;k++)
                    a[j,k] = a[j,k] – m * a[i,k];
                b[j] = b[j] – m_ji * b[i];
        }
    }
Step 7: if a[n,n] == 0 then output('no unique solution exists"); stop.
Step 8: //Start backward substitution
    for (i=n; i>0; i--){
        for (j=i+1;j<=n; j++)
            b[i] = b[i] – a[i,j] * x[j];
        x[i] = b[i] / a[i,i];
    }
STEP 9:
    OUTPUT(x[1],x[2],…,x[n]); //Procedure completed successfully
    STOP.
```

Example 2.2.3

The purpose of this example is to show what can happen if Algorithm 2.2.1 fails. The computations will be done simultaneously on two linear systems.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}, \qquad b_1 = \begin{bmatrix} 4 \\ 6 \\ 6 \end{bmatrix}, \qquad b_2 = \begin{bmatrix} 4 \\ 4 \\ 6 \end{bmatrix}$$

First, we construct augmented matrix:

$$A = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 1 & 6 \\ 1 & 1 & 2 & 6 \end{array}\right] \qquad \tilde{A} = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 1 & 4 \\ 1 & 1 & 2 & 6 \end{array}\right]$$

Thus, we have:

$$A = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 1 & 6 \\ 1 & 1 & 2 & 6 \end{array}\right] \xrightarrow[(E_3 - E_1) \to (E_3)]{(E_2 - 2E_1) \to (E_2)} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

$$\tilde{A} = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 1 & 4 \\ 1 & 1 & 2 & 6 \end{array}\right] \xrightarrow[(E_3 - E_1) \to (E_3)]{(E_2 - 2E_1) \to (E_2)} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 0 & -1 & -4 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

And then, the algorithm 2.2.1 requires that the procedure be halted, and no solution to either equation is obtained.

The first linear system has an infinite number of solutions; $x_3=2$, $x_2=2-x_1$, and x_1 arbitrary. The second system leads to the contradiction $x_3=2$ and $x_3=4$, so no solution exists. In each case, however, there is no unique solution, as we conclude from Algorithm 2.2.1.

Although Algorithm 2.2.1 can be viewed as the construction of the augmented matrices A, $A^{(1)}$, $A^{(2)}$, … , $A^{(n)}$, the computations can be performed in a computer using only one $n\times(n+1)$ array for storage. At each step we simply replace the previous value of a_{ij} by the new one. In addition, we can store the multipliers m_{ji} in the locations of a_{ji} since a_{ji} has the value 0 for each i=1,2,…,n-1 and j=i+1,i+2,…,n. Thus, *A* can be overwritten by the multipliers below the main diagonal and by the nonzero entries of $A^{(n)}$ on and above the main diagonal. These values can be used to solve other linear systems involving the original matrix A, as we will see in Section 2.3.

Both the amount of time required to complete the calculations

and the subsequent roundoff error depend on the number of floating-point arithmetic operations needed to solve a routine problem. In general, the amount of time required to perform a multiplication or division on a computer is approximately the same and is considerably greater than that required to perform an addition or subtraction. The actual differences in execution time, however, depend on the particular computing system. To demonstrate the counting operations for a given method, we will count the operations required to solve a typical linear equation of n equations in n unknows using Algorithm 2.2.1. We will keep the count of the additions/subtractions separate from the count of the multiplications/divisions because of the time differential.

No arithmetic operations are performed until Steps 5 and 6 in the algorithm. Step 5 requires that (n-i) divisions be performed. The replacement of the equation E_j by (E_j - $m_{ji}E_i$) in Step 6 requires that m_{ji} be multiplied by each term in E_i, resulting in a total of (n-i)(n-i+1) multiplications. After this is completed, each term of the resulting equation is subtracted from the corresponding term in E_j. This requires (n-i)(n-i+1) subtractions. For each i=1, 2, … , n-1, the operations required in Steps 5 and 6 are as follows.

Multiplications / divisions

(n-i)+(n-i)(n-i+1) = (n-i)(n-i+2).

Additions / subtractions

(n-i)(n-i+1)

The total number of operations required by these steps is obtained by summing the operating counts for each i. Recalling from calculus that

$$\sum_{j=1}^{m} 1 = m, \quad \sum_{j=1}^{m} j = \frac{m(m+1)}{2}, \quad \text{and} \quad \sum_{j=1}^{m} j^2 = \frac{m(m+1)(2m+1)}{6}$$

We have the following operation counts.

Multiplications/divisions

$$\sum_{i=1}^{n-1}(n-i)(n-i+2) = \sum_{i=1}^{n-1}(n^2-2ni+i^2+2n-2i) = \frac{2n^3+3n^2-5n}{6}$$

Additions/subtractions

$$\sum_{i=1}^{n-1}(n-i)(n-i+1) = \sum_{i=1}^{n-1}(n^2-2ni+i^2+n-i) = \frac{n^3-n}{3}.$$

The only other steps in Algorithm 2.2.1 that involve arithmetic operations are those required for backward substitution, Step 8 and 9, Step 8 requires one division. Step 9 requires (n-i) multiplications and (n-i-1) additions for each summation term and then one subtraction and one division. The total number of operations in Step 8 and 9 is as

follows.

Multiplications/divisions

$$1 + \sum_{i=1}^{n-1}((n-i)+1) = \frac{n^2+n}{2}$$

Additions/subtractions

$$\sum_{i=1}^{n-1}((n-i-1)+1) = \frac{n^2-n}{2}$$

The total number of arithmetic operations in Algorithm 2.2.1 is, therefore:

Multiplications/divisions

$$\frac{2n^3+3n^2-5n}{6} + \frac{n^2+n}{2} = \frac{n^3}{3} + n^2 - \frac{n}{3},$$

Additions/subtractions

$$\frac{n^3-n}{3} + \frac{n^2-n}{2} = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$$

For large n, the total number of multiplications and divisions is approximately $n^3/3$, as is the total number of additions and subtractions. Thus, the amount of computation and the time required increases with n in proportion to n^3.

§2.3 LU Decomposition and Its Applications

Suppose we are able to write the matrix A as a product of two matrices,

$$L \times U = A \qquad (2.3.1)$$

where L is lower triangular (has elements only on the diagonal and below) and U is upper triangular (has elements only on the diagonal and above). For the case of a 4 × 4 matrix A, for example, equation (2.3.1) would look like this:

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \times \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \qquad (2.3.2)$$

We can use a decomposition such as (2.3.1) to solve the linear set

$$\mathbf{A}\mathbf{x} = (\mathbf{LU})\mathbf{x} = \mathbf{L}(\mathbf{U}\mathbf{x}) = \mathbf{b} \qquad (2.3.3)$$

by first solving for the vector **y** such that

$$\mathbf{L}\mathbf{y} = \mathbf{b} \qquad (2.3.4)$$

and then solving

$$\mathbf{Ux} = \mathbf{y} \qquad (2.3.5)$$

What is the advantage of breaking up one linear set into two successive ones?

The advantage is that the solution of a triangular set of equations is quite trivial. Thus, equation (2.3.4) can be solved by forward substitution as follows:

$$y_1 = \frac{b_1}{\alpha_{11}}$$

$$y_i = \frac{1}{\alpha_{ii}}\left(b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j\right) \quad \text{i=2,3,…,n} \qquad (2.3.6)$$

While (2.3.5) can then be solved by backsubstitution as follows:

$$x_n = \frac{y_n}{\beta_{nn}}$$

$$x_i = \frac{1}{\beta_{ii}}\left(y_i - \sum_{j=i+1}^{n} \beta_{ij} x_j\right) \qquad \text{i=n-1,n-2,…,1} \qquad (2.3.7)$$

Equations (2.3.6) and (2.3.7) total (for each right-hand side **b**) N^2 executions of an inner loop containing one multiply and one add. If we have N right-hand sides that are the unit column vectors (which is the case when we are inverting a matrix), then taking into account the leading zeros reduces the total execution count of (2.3.6) from $n^3/2$ to $n^3/6$, while (2.3.7) is unchanged at $n^3/2$

Notice that, once we have the LU decomposition of A, we can solve with as many right-hand sides as we then care to, one at a time. This is a distinct advantage over the methods of §2.2.

Performing the LU Decomposition

How then can we solve for L and U, given A? We give an example.

Example 2.3.1 Gived a set of equations:

$$x_1 + 2x_2 + 3x_3 = 14$$
$$2x_1 + 5x_2 + 2x_3 = 18$$
$$3x_1 + x_2 + 5x_3 = 20$$

We construct a coefficient matrix A as follows:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \\ 3 & 1 & 5 \end{bmatrix}$$

And then, the sequence of operations $(E_2\text{-}2E_1)\rightarrow(E_2)$, $(E_3\text{-}3E_1)\rightarrow(E_3)$ and $(E_3\text{-}(\text{-}5)E_2)\rightarrow(E_3)$ converts the matrix to upper triangular matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \\ 3 & 1 & 5 \end{bmatrix} \xrightarrow[(E_3-3E_1)\to(E_3)]{(E_2-2E_1)\to(E_2)} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -4 \\ 0 & -5 & -4 \end{bmatrix} \xrightarrow{(E_3-(-5)E_2)\to(E_3)} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -4 \\ 0 & 0 & -24 \end{bmatrix}$$

The upper triangular matrix is matrix U. Lower triangular L can be given through the operations. That is: l_{21}=2(because of (E_2-2E_1)→(E_2)), l_{31}=3(because of (E_3-3E_1) →(E_3)), and l_{32}=-5(because of (E_3-(-5)E_2)→(E_3)). Therefore,

$$A=\begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \\ 3 & 1 & 5 \end{bmatrix}, \text{ then } L=\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -5 & 1 \end{bmatrix}, U=\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -4 \\ 0 & 0 & -24 \end{bmatrix}$$

LU Decomposition Algorithm

To decompose the n×n matrix A=(a_{ij}) into the product of the lower triangular matrix L=(l_{ij}) and the upper triangular matrix U=(u_{ij}); that is ,A = LU, where the main diagonal of either L or U consists of all ones:

Algorithm 2.3.1 LU decomposition algorithm

INPUT dimension n; the entries a_{ij}, 1≤i,j≤n of A; the diagonal $l_{11}=l_{22}=\ldots=l_{nn}=1$ of L or the diagonal $u_{11}=u_{22}=\ldots=u_{nn}=1$ of U.

OUTPUT the entries l_{ij},1≤j≤i,1≤i≤n of L and the entries, u_{ij}, i≤j≤n,1≤i≤n of U.

In the previous discussion we assumed that A**x** = **b** can be solved using Gaussian elimination without row interchanges. From a practical standpoint, this LU decomposition is useful only when row interchanges are not required to control the roundoff error resulting from the use of finite-digit arithmetic. Fortunately, many systems we encounter when using approximation methods are of this type, but we will now consider the modifications that must be made when row interchanges are required. We begin the discussion with the introduction of a class of matrices that are used to rearrange, or permute, rows of a given matrix.

An n×n **permutation** matrix P is obtained by rearranging the rows of I_n, the identity matrix. This gives a matrix with precisely one nonzero entry in each row and in each column, and each nonzero entry in A 1.

Example 2.3.2 Given matrix:

$$A = \begin{bmatrix} 0 & 1 & -1 & 1 \\ 1 & 1 & -1 & 2 \\ -1 & -1 & 1 & 0 \\ 1 & 2 & 0 & 2 \end{bmatrix}$$

Since $a_{11}=0$, the matrix A does not have an LU decomposition. However, using the row interchange $(E_1)\leftrightarrow(E_2)$, followed by $(E_3+E_1)\rightarrow(E_3)$ and $(E_4-E_1)\rightarrow(E_4)$, produces

$$\begin{bmatrix} 0 & 1 & -1 & 1 \\ 1 & 1 & -1 & 2 \\ -1 & -1 & 1 & 0 \\ 1 & 2 & 0 & 2 \end{bmatrix} \xrightarrow{(E_1)\leftrightarrow(E_2)} \begin{bmatrix} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & 1 \\ -1 & -1 & 1 & 0 \\ 1 & 2 & 0 & 2 \end{bmatrix} \xrightarrow[(E_4-E_1)\rightarrow(E_4)]{(E_3+E_1)\rightarrow(E_3)} \begin{bmatrix} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

, $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{(E_1)\leftrightarrow(E_2)} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

So,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & & 1 & 0 \\ 1 & & & 1 \end{bmatrix}$$

Then followed by $(E_4-E_2)\rightarrow(E_4)$,the row interchange $(E_3)\leftrightarrow(E_4)$, gives the matrix

$$\begin{bmatrix} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 1 & 1 & 0 \end{bmatrix} \xrightarrow{(E_4-E_2)\rightarrow(E_4)} \begin{bmatrix} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & -1 \end{bmatrix} \xrightarrow{(E_3)\leftrightarrow(E_4)} \begin{bmatrix} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{(E_3)\leftrightarrow(E_4)} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = P$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 1 & & 1 \end{bmatrix} \xrightarrow{(E_3)\leftrightarrow(E_4)} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

Gaussian elimination can be performed on PA without row interchanges to give the LU decomposition of PA.

$$PA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 2 \end{bmatrix} = LU$$

So, $\mathbf{Ax=b}$, $\leftrightarrow$ $\mathbf{PAx=Pb}$ $\leftrightarrow$ $\mathbf{LUx = Pb}$

Therefore, first solving for the vector $\mathbf{y}$ such that

$\mathbf{Ly = Pb}$

and then solving

$\mathbf{Ux = y}$

We can use the previous method to solve the problem.

§2.4 LU Decomposition with Pivot Element

In LU decomposition algorithm 2.3.1, if a_{kk} is very small, roundoff error introduced in the computation of one of the terms can be dramatically increase because of the division by a_{kk}.

Example 2.4.1

$$A = \begin{bmatrix} 0.01 & 1.1 & 3.1 \\ 2 & 4.5 & 0.36 \\ 9 & 0.96 & 6.5 \end{bmatrix}, \quad \mathrm{b} = \begin{bmatrix} 7.28 \\ 1.22 \\ -4.04 \end{bmatrix}$$

LU decomposition:

$$A = \begin{bmatrix} 0.01 & 1.1 & 3.1 \\ 2 & 4.5 & 0.36 \\ 9 & 0.96 & 6.5 \end{bmatrix} \xrightarrow[(E_3-900E_1)\to(E_3)]{(E_2-200E_1)\to(E_2)} \begin{bmatrix} 0.01 & 1.1 & 3.1 \\ 0 & -215.5 & -619.64 \\ 0 & -989.04 & -2783.5 \end{bmatrix}$$

$$\begin{bmatrix} 0.01 & 1.1 & 3.1 \\ 0 & -215.5 & -619.64 \\ 0 & -989.04 & -2783.5 \end{bmatrix} \xrightarrow{(E_3-4.59E_2)\to(E_3)} \begin{bmatrix} 0.01 & 1.1 & 3.1 \\ 0 & -215.5 & -619.64 \\ 0 & 0 & 60.648 \end{bmatrix} = U$$

So, $L = \begin{bmatrix} 1 & 0 & 0 \\ 200 & 1 & 0 \\ 900 & 4.59 & 1 \end{bmatrix}$

The exact solution of the equation $\mathbf{Ax=b}$ is $x_1=-2$, $x_2=1$,$x_3=2$. But, we use LU decomposition to solve this equation, and give the approximation solution: $x_1 \approx -1.9$, $x_2 \approx 0.994$, $x_3 \approx 2.002$.

This example illustrates that LU decomposition produce roundoff error.

The roundoff error may be avoided by LU Decompositon with Column Pivot Element .

Example 2.4.2

$$A=\begin{bmatrix}0.01 & 1.1 & 3.1\\ 2 & 4.5 & 0.36\\ 9 & 0.96 & 6.5\end{bmatrix},\quad \mathrm{b}=\begin{bmatrix}7.28\\ 1.22\\ -4.04\end{bmatrix}$$

LU Decompositon with Column Pivot Element is:

First, search the max value from the first column, the max value is 9. The row interchange $(E_3)\leftrightarrow(E_1)$, gives the matrix

$$A'=\begin{bmatrix}9 & 0.96 & 6.5\\ 2 & 4.5 & 0.36\\ 0.01 & 1.1 & 3.1\end{bmatrix}\qquad P=\begin{bmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}$$

So,

$$A'\xrightarrow[(E_3-0.001E_1)\to(E_3)]{(E_2-0.222E_1)\to(E_2)}\begin{bmatrix}9 & 0.96 & 6.5\\ 0 & 4.287 & -1.084\\ 0 & 1.099 & 3.093\end{bmatrix}\xrightarrow{(E_3-0.256E_2)\to(E_3)}$$

$$\begin{bmatrix}9 & 0.96 & 6.5\\ 0 & 4.287 & -1.084\\ 0 & 0 & 3.371\end{bmatrix}=U,\qquad L=\begin{bmatrix}1 & 0 & 0\\ 0.222 & 1 & 0\\ 0.001 & 0.256 & 1\end{bmatrix}$$

Therefore:

$\mathbf{Ax}=\mathbf{b}\quad\equiv\quad \mathbf{PAx}=\mathbf{Pb}\quad\equiv\quad \mathbf{LUx}=\mathbf{Pb}$

The solution of the equation $\mathbf{Ly}=\mathbf{Pb}=\begin{bmatrix}-4.04\\ 1.22\\ 7.28\end{bmatrix}$ is: $y_1\approx-4.04$, $y_2\approx2.117$, and $y_3\approx6.742$

The solution of the equation $\mathbf{Ux}=\mathbf{y}$ is : $x_1\approx-2$, $x_2\approx1$, and $x_3\approx2$.

§2.5 Norms of Vectors and Matrices

To discuss iterative methods for solving linear systems, we first need to be able to measure the distance between n-dimensional column vectors to determine whether a sequence of vectors converges to a solution of the systems. In actuality, this measure is also needed when the solution is obtained by the direct methods presented in §2.2. Those methods required a large number of arithmetic operations, and using finite-digit arithmetic leads only to an approximation to an actual solution of the system.

Let $\mathbf{R}^n$ denote the set of all n-dimensional column vectors with real-number coefficients. To define a distance in $\mathbf{R}^n$ we use the notion of a norm.

§2.5.1 Norms of Vectors

Definition 2.5.1 A **vector Norm** on $\mathbf{R}^n$ is a function, $||\cdot||$, from $\mathbf{R}^n$ into R with the following properties:

i. $||\mathbf{x}||\geq 0$ for all $\mathbf{x}\in\mathbf{R}^n$,
ii. $||\mathbf{x}||=0$ if and only if $\mathbf{x} = \mathbf{0}$,
iii. $||\alpha\mathbf{x}||=|\alpha|\ ||\mathbf{x}||$ for all $\alpha\in\mathbf{R}$ and $\mathbf{x}\in\mathbf{R}^n$,
iv. $||\mathbf{x}+\mathbf{y}||\leq||\mathbf{x}|| + ||\mathbf{y}||$ for all $\mathbf{x},\mathbf{y}\in\mathbf{R}^n$.

We will need only two specific norms on $\mathbf{R}^n$, although a third and more on $\mathbf{R}^n$ is presented.

Since vectors in $\mathbf{R}^n$ are column vectors, it is convenient to use the transpose notation when a vector is represented in terms of its components. For example, the vector

$$\mathbf{X}=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

will be written $\mathbf{X}=(x_1,x_2,\ldots,x_n)^T$.

Definition 2.5.2 The l_1 and l_∞ for the vector $\mathbf{X}=(x_1,x_2,\ldots,x_n)^T$ are defined by

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} | x_i | \quad \text{and} \quad ||\mathbf{x}||_\infty = \max_{1\leq i\leq n} \{| x_i |\}$$

Example 2.5.1

The vector $\mathbf{x}=(-1,1,-2)^T$ has norms

$||\mathbf{x}||_1 = 4$ and $||\mathbf{x}||_\infty = 2$

§2.5.2 Norms of Matrices

Definition 2.5.3 A **Matrix Norm** on the set of all n×n matrices is a real-value function, $||\cdot||$, defined on this set, satisfying for all n×n matrices A and B and all real numbers α:

1) $||A||\geq 0$;
2) $||A||=0$ if and only if A is ***0***, the matrix with all 0 entries;
3) $||\alpha A||=|\alpha|\ ||A||$;
4) $||A+B||\leq||A|| + ||B||$;
5) $||AB|| \leq||A||\ ||B||$.

Easily we can get $||A+B|| \geq |\ ||A|| - ||B||\ |$

Although matrix norms can be obtained in various ways, the only norms we consider are those that are natural consequences of the vector norms l_1 and l_∞.

Theorem 2.5.1 If $||\cdot||$ is a vector norm on R^n , then

$$||A|| = \max_{\|x\|=1} \| Ax \|$$

is a matrix norm.

Proof is omitted.

This is called the **natural**, or induced, **matrix norm** associated with the vector norm. In this text, all matrix norms will be assumed to be natural matrix norms unless specified otherwise.

For any $z \neq 0$, we have $x=z/||z||$ as a unit vector. Hence,

$$\max_{\|x\|=1} \| Ax \| = \max_{z\neq 0} \| A(\frac{z}{\|z\|}) \| = \max_{z\neq 0} \frac{\|Az\|}{\|z\|}$$

and we can alternatively write

$$\|A\| = \max_{\|x\|\neq 0} \frac{\| Ax \|}{\|x\|}$$

The following corollary to Theorem 2.5.1 follows from this representation of $\|A\|$.

Corollary 2.5.1 For any vector $x \neq 0$, and any natural norm $\|\cdot\|$, we have

$$\|Ax\| \leq \|A\| \, \|x\|$$

Theorem 2.5.2 If $A=(a_{ij})$ is an $n \times n$ matrix, then

$$\| A \|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} | a_{ij} | \qquad \text{(be called column norm of A)}$$

$$\| A \|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| \qquad \text{(be called row norm of A)}$$

Proof

① $$\| A \|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} | a_{ij} |$$

Let $x=(x_1,x_2,\ldots,x_n)^T \neq 0$. Since $Ax = (\sum_{j=1}^{n} a_{1j}x_j, \sum_{j=1}^{n} a_{2j}x_j, \cdots, \sum_{j=1}^{n} a_{nj}x_j)^T$ is also an n-dimensional vector, $\| Ax \|_1 = \sum_{i=1}^{n} | \sum_{j=1}^{n} a_{ij}x_j | \leq \sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}x_j| =$

$$\sum_{j=1}^{n} \sum_{i=1}^{n} |a_{ij}x_j| = \sum_{j=1}^{n} (|x_j| \sum_{i=1}^{n} |a_{ij}|) \qquad (2.5.1)$$

Let k be an integer with $\max_j \sum_{i=1}^{n} |a_{ij}| = \sum_{i=1}^{n} |a_{ik}|$, So,

$$\| Ax \|_1 \leq \sum_{j=1}^{n} (|x_j| \sum_{i=1}^{n} |a_{ij}|) \leq \sum_{j=1}^{n} (|x_j| \sum_{i=1}^{n} |a_{ik}|) = \sum_{j=1}^{n} |x_j| \sum_{i=1}^{n} |a_{ik}| = \|x\|_1 \|A\|_1$$

and **x** be the vector with components

$$x_j = \begin{cases} 0 & \text{if } j \neq k \\ 1 & \text{if } j = k \end{cases}$$

then $\| x \|_1 = 1$ and $\| Ax \|_1 = \sum_{i=1}^{n} | \sum_{j=1}^{n} a_{ij} x_j | = \sum_{i=1}^{n} |a_{ik}| = \max_j \sum_{i=1}^{n} |a_{ij}|$, so

$$\| A \|_1 = \max_{x \neq 0} \frac{\| Ax \|_1}{\| x \|_1}$$

② $\| A \|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$

Let $\mathbf{x}=(x_1,x_2,\ldots,x_n)^T \neq \mathbf{0}$, $A \neq \boldsymbol{0}$, and $t=\|x\|_\infty$, $v= \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$, so

$$\| Ax \|_\infty = \max_{1 \leq i \leq n} | \sum_{j=1}^{n} a_{ij} x_j | \leq \max_i \sum_{j=1}^{n} |a_{ij}| |x_j| \leq t \max_i | a_{ij} |$$

This show for all nonzero vector **x** , $\frac{\| Ax \|_\infty}{\| x \|_\infty} \leq v$

Now we let k be an integer with $\max_i \sum_{j=1}^{n} |a_{ij}| = \sum_{j=1}^{n} |a_{kj}|$

And x be the vector with components

$$x_j = \begin{cases} 1, & \text{if } a_{kj} \geq 0 \\ -1, & \text{if } a_{kj} < 0 \end{cases}$$

Then $\|\mathbf{x}\|_\infty = 1$ and $a_{kj}x_j = |a_{kj}|$, for all j=1,2,…,n, so

$$\| Ax \|_\infty = \max_{1 \leq i \leq n} | \sum_{j=1}^{n} a_{ij} x_j | \geq | \sum_{j=1}^{n} a_{kj} x_j | = \sum_{j=1}^{n} |a_{kj}| = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}| = v$$

This result implies that

$$\| Ax \|_\infty = \max_{1 \leq i \leq n} | \sum_{j=1}^{n} a_{ij} x_j | \geq | \sum_{j=1}^{n} a_{kj} x_j | = \sum_{j=1}^{n} |a_{kj}| = v$$

Example 2.5.2

If $A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & -1 \\ 5 & -1 & 1 \end{bmatrix}$, then $\|A\|_1 = 6$ and $\|A\|_\infty = 7$

§2.6 Error Analysis

It seems intuitively reasonable that if **x'** is an approximation to the solution **x** of A**x**=**b** and the residual vector **r**=**b** – A**x'** has the property thar ||**r**||=||**b**-A**x'**|| is small, then ||**x**-**x'**|| would be small as well. This is often the case, but certain systems, which occur frequently in practice, fail to have this property.

Example 2.6.1

The linear system A**x**=**b** given by

$$\begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Has the unique solution $\mathbf{x}=(2,0)^T$.

When b has a small error, $\mathbf{b}=(2,2.0001)^T$, the unique solution is $\mathbf{x'}=(1,1)^T$.

The example was clearly constructed to show the difficulties that can --and, in fact, do --- arise. Had the lines not been nearly coincident, we would expect a small residual vector to imply an accurate approximation.

In the general situation, we cannot rely on the geometry of the system to give an indication of when problems might occur. We can , however, obtain this information by considering the norms of the matrix A and its inverse.

Definition 2.6.1 Let **x'** be an approximation to the solution of A**x**=**b**, then **r**=**b** – A**x'** is called as the residual vector, and ||**r**||=||**b** – A**x'**|| is called as backward error, ||**x** - **x'**|| is called as forward error.

Theorem 2.6.1 Suppose that **x'** be an approximation to the solution of A**x**=**b**, A is a nonsingular matrix, and **r** is the residual vector for **x'**. Then for any natural norm,

$$||\mathbf{x} - \mathbf{x'}|| \le ||\mathbf{r}|| \, ||A^{-1}||$$

And if **x** ≠ **0** and **b** ≠ **0**,

$$\frac{||x-x'||}{||x||} \le ||\, A \,|| \bullet ||\, A^{-1} \,|| \, \frac{||r||}{||b||} \qquad (2.6.1)$$

Proof Since **r** = **b** – A**x'** = A**x** – A**x'** and A is nonsingular, **x** – **x'** = A^{-1}**r**. Theorem 2.5.1 implies that

$$||\mathbf{x} - \mathbf{x'}|| = ||A^{-1}\mathbf{r}|| \le ||A^{-1}|| \, ||\mathbf{r}||.$$

Moreover, since **b** = A**x**, we have ||**b**|| ≤ ||A|| ||**x**||, so 1/||**x**|| ≤ ||A|| / ||**b**|| and

$$\frac{||x-x'||}{||x||} \le ||\, A \,|| \bullet ||\, A^{-1} \,|| \, \frac{||r||}{||b||}$$

The inequalities in Theorem 2.6.1 imply that $||A^{-1}||$ and ||A|| $||A^{-1}||$ provide an indication of the connection between the residual vector and the accuracy of the approximation. In general, the relative error ||**x** – **x'**|| / ||**x**|| is of most interest, and, by Inequality (2.6.1), this error is bounded by the product of ||A|| $||A^{-1}||$ with the relative residual for this approximation. ||**r**|| / ||**b**||. Any convenient norm can be used for this approximation; the only requirement is that it be used consistently throughout.

Definition 2.6.2 The **Condition number** of the nonsingular matrix A relative to a

norm || • || is

$$\text{cond}(A) = ||A|| \cdot ||A^{-1}||.$$

For any nonsingular matrix A and natural norm || • ||,

$$1 = ||I|| = ||A \cdot A^{-1}|| \le ||A|| \cdot ||A^{-1}|| = \text{cond}(A)$$

A matrix A is **well-conditioned** if cond(A) is close to 1, and is **ill-conditioned** when cond(A) is significantly greater than 1. Conditioning in this context refers to the relative security that a small residual vector implies a correspondingly accurate approximation solution.

Example 2.6.2

The matrix for the system considered in example 2.6.1 was

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix}, \quad \text{then} \quad A^{-1} = \begin{bmatrix} 10001 & -10000 \\ -10000 & 10000 \end{bmatrix}$$

Which has $||A||_1 = 2.0001$, and $||A^{-1}||_1 = 20001$

For the 1 norm, cond(A) = (2.0001)(20001) = 40004.0001. The size of the condition number for this example should certainly keep us from making hasty accuracy decisions based on the residual of an approximation.

§2.7 Iterative Techniques for Solving Linear Systems

In this section we describe the Jacobi and the Gauss-Seidel iterative methods, Classic methods that date to the late eighteenth century. Iterative techniques are seldom used for solving linear systems of small dimension since the time required for sufficient accuracy exceeds that required for direct techniques such as Gaussian elimination. For large systems with a high percentage of 0 entries, however, these techniques are efficient in terms of both computer storage and computation. Systems of this type arise frequently in circuit analysis and in the numerical solution of boundary-value problems and partial-differential equations.

An iterative technique to solve the n × n linear system A**x** = **b** starts with an initial approximation $\mathbf{x}^{(0)}$ to the solution **x** and generates a sequence of vector $\{x^{(k)}\}_{k=0}^{\infty}$ that converges to **x**. Iterative techniques involve a process that converts the system A**x** = **b** into an equivalent system of the form **x** = T**x** + **c** for some fixed matrix T and vector **c**.

After the initial vector $\mathbf{x}^{(0)}$ is selected, the sequence of approximation solution vectors is generated by computing.

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}$$

for each k = 1,2, … .

§2.7.1 Jacobi iterative method

Example 2.7.1 The linear system A**x** = **b** given by

E_1: 3x + y = 5

E_2: x + 2y = 5

Has the unique solution x = 1, y = 2. To convert A**x** = **b** to the form **x** = T**x** +

c, we have two method to do it.

Method 1 : Solve equation E_i for x_i , for each i = 1,2, …, to obtain

x = (5 – y) / 3

y = (5 – x) / 2

For an initial approximation, we let $(x^{(0)},y^{(0)})^T = (0,0)^T$. Then $(x^{(1)},y^{(1)})^T = (5/3,5/2)^T$, $(x^{(2)},y^{(2)})^T = (5/6,5/3)^T$, … $(1,2)^T$.

Method 2: Use following form for x_i

x = 5 – 2y

y = 5 -3x

For an initial approximation, we let $(x^{(0)},y^{(0)})^T = (0,0)^T$. Then $(x^{(1)},y^{(1)})^T = (5,5)^T$, $(x^{(2)},y^{(2)})^T = (-5,-10)^T$, $(x^{(3)},y^{(3)})^T = (25,20)^T$ ….

Jacobi iterative method consists of solving the ith equation in A**x** = **b** for x_i to obtain (provided $a_{ii} \neq 0$)

$$x_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(-\frac{a_{ij}x_j}{a_{ii}}\right) + \frac{b_i}{a_{ii}}, \text{ for i=1,2,…,n}$$

And generating each $x_i^{(k)}$ from components of $\mathbf{x}^{(k-1)}$ for k≥1 by

$$x_i^{(k)} = \frac{\sum_{\substack{j=1 \\ j \neq i}}^{n} (-a_{ij}x_j^{(k-1)} + b_i}{a_{ii}}, \text{ for i = 1,2,…,n}$$

The method is written in the form $\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}$ by splitting A into its diagonal and off-diagonal parts. To see this, let D be the diagonal matrix whose diagonal entries are those of A, -L be the strictly lower-triangular part of A, and -U be the strictly upper-triangular part of A. With this notation, matrix A is split into A = D – L – U.

The equation A**x** = **b,** or (D – L – U)**x** = **b**, is then transformed into

D**x** = (L + U)**x** + **b**,

and, if D^{-1} exists, that is, if $a_{ii} \neq 0$ for each i, then

$$\mathbf{x} = D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}. \qquad (2.7.1)$$

This results in the matrix form of the Jacobi iterative technique:

$$\mathbf{x}^{(k)} = D^{-1}(L + U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b}. \quad k = 1,2,\ldots \qquad (2.7.2)$$

Introducing the notation $T_j = D^{-1}(L+U)$ and $\mathbf{c}_j = D^{-1}\mathbf{b}$, the Jacobi technique has the form

$$\mathbf{x}^{(k)} = T_j\mathbf{x}^{(k-1)} + \mathbf{c}_j$$

§2.7.2 Gauss-Seidel iterative method

In (2.7.2), the components of $\mathbf{x}^{(k-1)}$ are used to compute $x_i^{(k)}$. Since, for i>1, $x_1^{(k)}$, $x_2^{(k)}$,…, $x_{i-1}^{(k)}$ have already been computed and are probably better approximations to the actual solutions x_1, x_2,…, x_{i-1} than $x_1^{(k-1)}$, $x_2^{(k-1)}$,…, $x_{i-1}^{(k-1)}$, it seems more reasonable to compute $x_i^{(k)}$ using these most recently calculated values. That is, we can use

$$x_i^{(k)} = \frac{-\sum_{j=1}^{i-1}(a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n}(a_{ij}x_j^{(k-1)} + b_i}{a_{ii}} \qquad (2.7.3)$$

for each i = 1,2,…,n, instead of Eq. (2.7.2). This modification is called the **Gauss-Seidel iterative technique** and is illustrated in the following example.

Example 2.7.2 Repeat example 2.7.1 use **Gauss-Seidel iterative technique.**

Solve equation E_i for x_i , for each i = 1,2, …, to obtain

$x = (5 – y) / 3$

$y = (5 – x) / 2$

So, $x^{(k)} = (5-y^{(k-1)}) / 3$

$y^{(k)} = (5 – x^{(k)}) / 2$

For an initial approximation, we let $(x^{(0)},y^{(0)})^T = (0,0)^T$. Then $(x^{(1)},y^{(1)})^T = (5/3,5/3)^T$, $(x^{(2)},y^{(2)})^T = (10/9,35/18)^T$, … $(1,2)^T$.

Since $\mathbf{x} = D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}$, so

$\mathbf{x} = D^{-1}(L\,\mathbf{x} + U\mathbf{x} + \mathbf{b})$

This results in the matrix form of the Gauss-Seidel iterative technique:

$$\mathbf{x}^{(k)} = D^{-1}(L\,\mathbf{x}^{(k)} + U\mathbf{x}^{(k-1)} + \mathbf{b}). \qquad k = 1,2,\ldots \qquad (2.7.4)$$

§2.7.3 Sufficiency Conditions for Convergence of the Jacobi and Gauss-Seidel methods

We can now give easily verified sufficiency conditions for convergence of the Jacobi and Gauss-Seidel methods.

Definition 2.7.1 If $A=(a_{ij})$, and $|a_{ii}| > \sum_{j\neq i}|a_{ij}|$, then we call A as **strictly diagonally dominant**.

Theorem 2.7.1 If A is a **strictly diagonally dominant,** then

2. A is a nonsingular matrix.
3. for any **b** and any choice of $\mathbf{x}^{(0)}$, both the Jacobi and Gauss-Seidel methods give sequences that converge to the unique solution of A**x** = **b.**

Proof.

1. Suppose that A is singular matrix, then the determinant of A is 0, or |A| = 0. That is equation A**x** = **0** with solution of nonzero **x'**. So, there is k, such that

$$|x_k| = \max_i\{|x_i|\} \neq 0$$

Then, through *k*th equation $\sum_{j=1}^{n} a_{kj}x_j = 0$, we have

$$|a_{kk}x_k| = \left|\sum_{\substack{j=1 \\ j\neq k}}^{n} a_{kj}x_j\right| \le \sum_{j\neq k}|a_{kj}x_j| \le |x_k| \sum_{j\neq k}|a_{kj}|$$

Hence, $|a_{kk}| \le \sum_{j\neq k}|a_{kj}|$.

This is impossible because A is a **strictly diagonally dominant.**

2. This proof need to use eigenvalue and eigenvector.

§2.7.4 Successive Over-Relaxation Methods.

The equation A**x** = **b**, or (D - L - U)**x** = **b**, is multiplied by the ω in both sides of the equation.

ω(D - L - U)**x** =ω **b** or (D-ωL)**x** =ω**b** + (1-ω)D**x** +ωU**x**

Therefore, iterative method is

$$(D-\omega L)\mathbf{x}^{(k)} = \omega\mathbf{b} + (1-\omega)D\mathbf{x}^{(k-1)} + \omega U\mathbf{x}^{(k-1)} \qquad (2.7.5)$$

or

$$\mathbf{x}^{(k)} = (D-\omega L)^{-1}[(1-\omega)D+\omega U]\,\mathbf{x}^{(k-1)} + \omega(D-\omega L)^{-1}\mathbf{b} \qquad (2.7.6)$$

For certain choices of positive ω, we can obtain significantly faster convergence.

For choices of ω with 0<ω<1, the procedure are called **under-relaxation methods** and can be used to obtain convergence of some systems that are not convergent by the Gauss-Seidel method. For choices of ω with 1 <ω, the procedures are called **over-relaxation methods**, which are used to accelerate the convergence for systems that are convergent by the Gauss-Seidel technique. These methods are abbreviated **SOR**, fro **Successive Over-Relaxation**, and are particularly useful for solving the linear systems that occur in the numerical solution of certain partial-differential equations.

Before illustrating the advantages of the SOR method, we see following example.

Example 2.7.3

The linear system A**x** = **b** given by

$$\begin{aligned} 4x_1 + 3x_2 &= 24 \\ 3x_1 + 4x_2 - x_3 &= 30 \\ -x_2 + 4x_3 &= -24 \end{aligned}$$

has the solution $(3,4,-5)^T$. The Gauss-Seidel method and the **SOR** method with ω = 1.25 will be used to solve this system, using $\mathbf{x}^{(0)} = (1,1,1)^T$ for both methods. For each k = 1,2, … , the equations for the Gauss-Seidel method are

$$x_1^{(k)} = -0.75x_2^{(k-1)} + 6$$
$$x_2^{(k)} = -0.75x_1^{(k)} + 0.25x_3^{(k-1)} + 7.5$$
$$x_3^{(k)} = 0.25x_2^{(k)} - 6,$$

and the equations for the SOR method with ω = 1.25 are

$$x_1^{(k)} = -0.25\,x_1^{(k-1)} - 0.9375x_2^{(k-1)} + 7.5$$

$x_2^{(k)}$=-0.9375$x_1^{(k)}$- 0.25 $x_2^{(k-1)}$+0.3125$x_3^{(k-1)}$+9.375,
$x_3^{(k)}$= 0.3125$x_2^{(k)}$ -0.25$x_3^{(k-1)}$-7.5.

Table 2.7.1 Gauss-Seidel

k	0	1	2	3	4	5	6	7
$x_1^{(k)}$	1	5.25	3.1406	3.0879	3.0549	3.0343	3.0215	3.0134
$x_2^{(k)}$	1	3.8125	3.8828	3.9268	3.9542	3.9714	3.9821	3.9888
$x_3^{(k)}$	1	-5.0469	-5.0293	-5.0183	-5.0114	-5.0072	-5.0045	-5.0028

Table 2.7.2 SOR with ω = 1.25

k	0	1	2	3	4	5	6	7
$x_1^{(k)}$	1	6.3125	2.6223	3.1333	2.9571	3.0037	2.9963	3.0000
$x_2^{(k)}$	1	3.5195	3.9585	4.0102	4.0074	4.0029	4.0009	4.0003
$x_3^{(k)}$	1	-6.6501	-4.6004	-5.0967	-4.9735	-5.0057	-4.9983	-5.0003

The first seven iterates for each method are listed in Table. For the iterates to be accurate to seven decimal places, the Gauss-Seidel method requires 34 iterations, as opposed to 14 iterations for the over-relaxation method with ω = 1.25.

The obvious question to ask is how the appropriate value of ω is chosen.

Exercise

1. Use Gaussian elimination with backward substitution and two-digit rounding arithmetic to solve the following linear systems. Do not reorder the equations.
   a. $4x_1 - x_2 + x_3 = 8$, $2x_1 + 5x_2 + 2x_3 = 3$, $x_1 + 2x_2 + 4x_3 = 11$.
   b. $4x_1 + x_2 + 2x_3 = 9$, $2x_1 + 4x_2 - x_3 = -5$, $x_1 + x_2 - 3x_3 = -9$.
2. Use LU decomposition to solve the following linear systems, if possible, and determine whether row interchanges are necessary:
   a. $x_1 - x_2 + 3x_3 = 2$, $3x_1 - 3x_2 + x_3 = -1$, $x_1 + x_2 = 3$.
   b. $2x_1 + 1.5x_2 + 3x_3 = 1$, $-x_1 + 2x_3 = 3$, $4x_1 - 4.5x_2 + 5x_3 = 1$.
3. Repeat Exercise 2 Use LU decomposition with pivot element.
4. Find $\|\cdot\|_1$ and $\|\cdot\|_\infty$ for the following matrices.
   a. $\begin{bmatrix} 10 & 15 \\ 0 & 1 \end{bmatrix}$ **b.** $\begin{bmatrix} 10 & 0 \\ 15 & 1 \end{bmatrix}$
   c. $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ **d.** $\begin{bmatrix} 4 & -1 & 7 \\ -1 & 4 & 0 \\ -7 & 0 & 4 \end{bmatrix}$
5. Compute the condition numbers of the following matrices to $\|\cdot\|_1$

a. $\begin{bmatrix} \frac{1}{2} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{4} \end{bmatrix}$ **b.** $\begin{bmatrix} 3.9 & 1.6 \\ 6.8 & 2.9 \end{bmatrix}$

c. $\begin{bmatrix} 1 & -1 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$ **d.** $\begin{bmatrix} 0.04 & 0.01 & -0.01 \\ 0.2 & 0.5 & -0.2 \\ 1 & 2 & 4 \end{bmatrix}$

6. The linear system A**x** = **b** given by

E_1: $10x_1 - x_2 + 2x_3 = 6$
E_2: $-x_1 + 11x_2 - x_3 + 3x_4 = 25$
E_3: $2x_1 - x_2 + 10x_3 - x_4 = -11$
E_4: $3x_2 - x_3 + 8x_4 = 15$

has the unique solution $\mathbf{x} = (1,2,-1,1)^T$. Please use Jacobi iterative method to get the solution.

7. Repeat exercise 6, use Gauss-Seidel iterative method to get the solution.
8. Find the first two iterations of the SOR method with $\omega = 1.1$ for the following linear systems using $\mathbf{x}^{(0)} = \mathbf{0}$:

a. $3x_1 - x_2 + x_3 = 1$
$3x_1 + 6x_2 + 2x_3 = 0,$
$3x_1 + 3x_2 + 7x_3 = 4,$

b. $10x_1 - x_2 = 9$
$-x_1 + 10x_2 - 2x_3 = 7$
$-2x_2 + 10x_3 = 6.$