Math 151a weeks 2 through 5 discussion notes

Jean-Michel Maldague
March 23, 2018

1. Order of convergence
2. Types of errors
3. Bisection method
4. Regula-Falsi method
5. Newton’s method
6. Secant method
7. General fixed-point method
8. Accelerating convergence
9. Computer arithmetic
10. MATLAB/octave demos

1 Order of convergence

Suppose you’re using a method to solve a root-finding or fixed-point problem. Let $x_k$ denote the kth iterate, $x^*$ be an actual root, and $e_k = x_k - x^*$ denote the root error at the kth step. If the error satisfies $\lim_{k \to \infty} e_k = 0$, $|e_{k+1}| \approx \lambda |e_k|^\alpha$ (i.e. $\lim_{k \to \infty} |e_{k+1}|/|e_k|^\alpha = \lambda$), where $\alpha \geq 1$, $\lambda > 0$, we say the method is order $\alpha$ with asymptotic error constant $\lambda$.

For methods where an upper bound on the error is easier to analyze than the true root error (such as bisection), we allow $e_k$ to denote our upper bound on the root error instead. We also allow $e_k = x^* - x_k$ and $e_k = |x_k - x^*|$ to be used as alternate versions of the root error.

When computing the order of a method, we generally assume as many (continuous) derivatives as we need, for all functions involved. Warning: never do this in an analysis class!

Note that: $\log \left( \frac{|e_{k+1}|}{|e_k|} \right) \approx \log \left( \frac{\lambda |e_k|^\alpha}{\lambda |e_{k-1}|^\alpha} \right) = \alpha \log \left( \frac{|e_k|}{|e_{k-1}|} \right)$. With this in mind, we can experimentally compute the approximate order of convergence and asymptotic error constant as: $\alpha \approx \frac{\log \left( |e_{k+1}|/|e_k| \right)}{\log \left( |e_{k+1}|/|e_{k-1}| \right)}$, $\lambda \approx \frac{|e_{k+1}|}{|e_k|^\alpha}$ (can fail if $\alpha = \lambda = 1$). More rigorously,

$$\frac{\log \left( |e_{k+1}|/|e_k| \right)}{\log \left( |e_k|/|e_{k-1}| \right)} = \frac{(1/\alpha) \log \left( |e_{k+1}|/|e_k|^{\alpha} \right) + (\alpha - 1) \log \left( |e_k| \right)}{\log \left( |e_k|/|e_{k-1}|^{\alpha} \right) + (\alpha - 1) \log \left( |e_k| \right)} \rightarrow \alpha \text{ as } k \rightarrow \infty.$$
2 Types of errors

With all these errors, we may consider them with or without absolute values/magnitudes (better to think of them as magnitudes).

**root error:** $e_k = |x_k - x^*|$, distance to actual root

**difference between iterates:** $|x_k - x_{k-1}|$, exactly as it sounds

**residual:** If we are trying to find the root of $f(x) = 0$, the residual is $|f(x_k)|$. It is a measure of how well our iterate “behaves as a root.”

The MVT gives us a $\xi$ between $x_k$ and $x^*$ such that $\frac{f(x_k) - f(x^*)}{x_k - x^*} = f'(\xi)$, which then gives us a nice relationship between the root error and the residual (independent of the root-finding method employed): $|f(x_k)| = |f'(\xi)||x_k - x^*|$. This helps us determine stopping conditions based on the residual, even when we don’t know the root (assuming $f$ has a continuous first derivative). In the case of Newton’s method, this condition can be rephrased in terms of a difference between successive iterates, as we will see later.
3 Bisection Method

This root-finding method applied to a continuous function $f$ starts with an initial interval $[a_0, b_0]$ such that $f(a_0) \cdot f(b_0) < 0$. A guess $x_0 = \frac{a_0 + b_0}{2}$ is ventured, and for the first halving of the interval, we select the half-interval over which the function has different signs at the endpoints. We continue in this manner, successively halving the intervals, thereby generating a sequence of closed, nested, nonempty intervals $[a_k, b_k]$ for nonnegative $k$. The iterates $x_k$, which are the midpoints of the corresponding intervals $[a_k, b_k]$, must converge to the unique real number $x^* \in \cap [a_k, b_k]$ (e.g. $\{a_k\}_{k=0}^\infty$ is a bounded & increasing sequence in $\mathbb{R}$ whose limit must be the unique point in the intersection). The IVT applied to each of the intervals guarantees the existence of some root, say, $w_k \in [a_k, b_k]$. Then also $w_k \to x^*$ as $k \to \infty$ due to the shrinking nature of the intervals, and then by continuity of $f$ we get that $x^*$ is an actual root. So the bisection method works.

Specifically, there is a nice formula for the root errors:

$$|x_k - x^*| \leq \frac{b_k - a_k}{2} = \ldots = \frac{b_0 - a_0}{2^k + 1}.$$

Because our (convenient) upper bound on the root errors exactly halves after each iteration, this method is considered order 1 ("linear convergence"), with asymptotic error constant $\lambda = \frac{1}{2}$. The bisection method is the "slow but sure" method that works when all else fails.
4 Regula-Falsi method

Same as bisection method, but instead of selecting the iterates $x_k$ to be the midpoints of the respective intervals $[a_k, b_k]$, we select them to be the roots of the secant lines of $f$ over the intervals $[a_k, b_k]$. The secant lines over $[a_k, b_k]$ can be represented as lines by using point-slope form: $y - f(a_k) = \left( \frac{f(b_k) - f(a_k)}{b_k - a_k} \right)(x - a_k)$. Then the root of this line is given by $x_k = a_k - \frac{f(a_k)(b_k - a_k)}{f(b_k) - f(a_k)} = \frac{a_k \cdot f(b_k) - b_k \cdot f(a_k)}{f(b_k) - f(a_k)}$. It is important to note that when you implement this into code, you should not use the interval widths as a stopping condition, since those might not shrink to 0, even as the root error, difference between iterates, and residual all do shrink to 0.

The method can be shown to converge of order 1, though it is a rather tedious proof involving cases. By the nature of the algorithm, the method works extremely well when $f$ is linear (affine) near the root. But if the function has severe concavity near the root, Regula-Falsi may be much slower than bisection, as the following graph shows.
5 Newton’s method

Suppose we are trying to find the root of a differentiable function $f : \mathbb{R} \to \mathbb{R}$. Newton’s method involves starting with a given initial iterate $x_0$ and selecting the next iterate $x_1$ to be the root of the tangent line to $f$ at the point $(x_0, f(x_0))$. $x_2$ is selected to be the root of the tangent line to $f$ at the point $(x_1, f(x_1))$, and so on and so forth. The tangent line to $f$ at $(x_k, f(x_k))$ is given by $y = f(x_k) + f'(x_k)(x - x_k)$, from which we obtain the root (and next iterate) as $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$. Clearly, if $f'(x_k) = 0$, the method fails. Geometrically, this corresponds to the case of a flat tangent line, which has no roots.

To analyze the order of convergence of this method, we use Taylor’s approximation theorem:

$$e_{k+1} = x_{k+1} - x^* = x_k - x^* - \frac{f(x_k)}{f'(x_k)} = x_k - x^* + \frac{f(x^*) - f(x_k)}{f'(x_k)}$$

$$= x_k - x^* + \frac{f'(x_k)(x^* - x_k) + \frac{f''(\xi)(x^* - x_k)^2}{2}}{f'(x_k)} = \frac{f''(\xi)}{2f'(x_k)}(x^* - x_k)^2$$

$$= \frac{f''(\xi)}{2f'(x_k)}e_k^2.$$

So if $f''$ exists near $x^*$ and $f'(x^*) \neq 0$, and $e_0$ is sufficiently small to begin with, then we can assert that Newton’s method converges of order $\alpha = 2$ (“quadratic convergence”), with asymptotic error constant $\lambda = \frac{f''(x^*)}{2f'(x^*)}$. If any of these conditions is not met, the errors might go down merely linearly, or not at all, as the following examples demonstrate. (Note: if $f''(x^*) = 0$, we will get super-quadratic convergence).
Suppose we are trying to find the root of \( f(x) = x^2 \), and \( x_0 \neq 0 \). Then \( x_{k+1} = x_k - \frac{x_k^2}{2x_k} = \frac{x_k}{2} \).

Since \( x^* = 0 \), we get that \( e_{k+1} = \frac{x_{k+1}}{x_k} = \frac{e_k}{2} \), so the convergence is linear. The problem here was that \( f'(x^*) = 0 \).

Suppose we are trying to find the root of \( f(x) = x^{1/3} \), and \( x_0 \neq 0 \). Then

\[
x_{k+1} = x_k - \frac{1}{3} x_k^{1/3} x_k^{-2/3} = x_k - 3x_k = -2x_k,
\]

and so \( e_{k+1} = -2e_k \), so in fact the method doesn’t converge at all. The problem here is that \( f''(x^*) \) blows up (faster than \( f' \)) near the root.
Suppose we are trying to find the root of \( f(x) = (x - 2)^2 - 1 \) with initial iterate \( x_0 = 2 \). Then \( x_{k+1} = x_k - \frac{(x_k - 2)^2 - 1}{2(x_k - 2)} \), so \( x_1 = 2 - \frac{(2 - 2)^2 - 1}{2(2 - 2)} = 2 - \frac{-1}{0} \), an absurdity. The problem here was that \( x_0 \) was not sufficiently close to \( x^* \).

And as the following graph shows, sometimes it’s not so obvious where \( x_0 \) should lie to ensure eventual convergence to the root. The theory only guarantees convergence if \( x_0 \) lies in some (potentially small) neighborhood of the root, \( x^* \). Below, \( x_0 \in (-1/2, 1/2) \) is required.

And so we may view Newton’s method as a powerful weapon for attacking root-finding problems, that must be handled with care. Bisection method is often used as a “starter” for Newton’s method to get the initial iterates close enough to the root. If \( f \) has a zero of order \( m < \infty \) at the root, we can apply Newton’s method to \( \frac{f}{f'} \) to regain quadratic convergence, but the implementation of this often causes significant errors due to roundoff. If \( f'' \) is unbounded near \( x^* \), this can cause serious problems for Newton’s method.
6 Secant method

The same as Newton’s method, but with derivatives approximated by finite differences instead. In the formula for $x_{k+1}$, the derivative $f'(x_k)$ is replaced by $\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$. In other words, the iteration for secant method is given by:

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

which is just the Regula-Falsi iteration in disguise. However, there is one important difference between the two: secant method always updates both the “a” and “b” values to be the latest iterates, whereas Regula-Falsi can cause one of those endpoints to lag behind. The two methods are similar only in that they involve secants, and the secant method should be thought of as a variation of Newton’s method rather than a variation of Regula-Falsi.

Note that we must “look back” two steps in order to calculate $x_{k+1}$. We need two initial iterates, $x_0$ and $x_1$, to run the procedure. These kinds of methods are known as multi-step methods, and they generally have more complicated error analyses than their single-step counterparts. The order calculation follows. We assume $f''$ exists in a neighborhood of $x_0$ and is continuous at $x_0$, $f'(x^*) \neq 0$, and $x_0$ is sufficiently close to $x^*$. In addition to frequent usage of Taylor’s error formula, a key ingredient in this calculation is: if $F(x) = \frac{f(x)}{x - x^*}$, then $F'(x) = \frac{f'(x)(x - x^*) - f(x)}{(x - x^*)^2}$, so by MVT, we may write

$$\frac{f(x_k)}{x_k - x^*} - \frac{f(x_{k-1})}{x_{k-1} - x^*} = \frac{F(x_k) - F(x_{k-1})}{x_k - x_{k-1}} = F'(\xi) = \frac{f'(\xi)(\xi - x^*) - f(\xi)}{(\xi - x^*)^2}.$$

8
Now let’s proceed with the order calculation.

\[ e_{k+1} = x_{k+1} - x^* \]

\[ = \frac{x_{k-1}f(x_k) - x_kf(x_{k-1})}{f(x_k) - f(x_{k-1})} - x^* \]

\[ = \frac{(x_{k-1} - x^*)f(x_k) - (x_k - x^*)f(x_{k-1})}{f(x_k) - f(x_{k-1})} \]

\[ = \frac{f(x_k) - f(x_{k-1})}{f(x_k) - f(x_{k-1})} \]

\[ = e_{k}e_{k-1} \frac{x_k - x_{k-1}}{x_k - x^*} \frac{x_k - x^*}{x_k - x_{k-1}} \]

\[ = \frac{1}{f'(\xi)} e_{k}e_{k-1} \frac{f'(\xi)(\xi - x^*) - f(\xi)}{(\xi - x^*)^2} \]

\[ = \frac{1}{f'(\xi)} e_{k}e_{k-1} \frac{f'(\xi)(\xi - x^*) - \left[f(x^*) + f'(x^*)(\xi - x^*) + \frac{f''(\xi)}{2}(\xi - x^*)^2\right]}{(\xi - x^*)^2} \]

\[ = -\frac{f''(\xi)}{2f'(\xi)} e_{k}e_{k-1}. \]

If we choose \( x_0 \) and \( x_1 \) close enough to \( x^* \), we can ensure that \( e_k \to 0 \) as \( k \to \infty \). Then we can write \( |e_{k+1}| = c_k|e_k||e_{k-1}| \), where \( c_k \to c := \left| -\frac{f''(x^*)}{2f'(x^*)} \right| \) by continuity. If any \( e_k = 0 \), then we may stop the algorithm, as we have found an exact root of \( f \). Otherwise, define \( p = \frac{1 + \sqrt{5}}{2} \) (the golden ratio), noting that \( p \) satisfies \( p^2 = p + 1 \). Put \( y_k = \frac{|e_{k+1}|}{|e_k|^p} \), and note that \( y_k = \frac{c_k|e_k||e_{k-1}|}{|e_k|^p} = \frac{c_k|e_{k-1}|}{|e_k|^{p-1}} = c_k \left( \frac{|e_{k-1}|}{|e_k|^{p-1}} \right)^{1/p} = c_k \left( \frac{|e_{k-1}|}{|e_k|} \right)^{1/p} = c_ky_{k-1}. \)

Now suppose \( f''(x^*) \neq 0 \). If any \( c_k = 0 \), then we would’ve obtained \( e_{k+1} = 0 \), a case we already considered. Define \( z_k = \log (y_k) \), \( a_k = \log (c_k) \), so that

\[ z_k = a_k + \left( \frac{-1}{p} \right) z_{k-1} = a_k + \left( \frac{-1}{p} \right) \left( a_{k-1} + \left( \frac{-1}{p} \right) z_{k-2} \right) = \cdots = z_0 \left( \frac{-1}{p} \right)^k + S_k, \]

where \( S_k = \sum_{j=0}^{k-1} a_{k-j} \left( \frac{-1}{p} \right)^j \). Set \( a = \lim a_k = \log (c) \), and note that

\[ \sum_{j=0}^{\infty} \left( \frac{-1}{p} \right)^j \frac{p}{p+1} \quad \text{and} \quad \sum_{j=0}^{\infty} \frac{1}{p^j} = \frac{p}{p-1} \quad \text{are geometric series.} \]
Let \( N \in \mathbb{N} \). For \( k > N \), we get

\[
\left| S_k - a \sum_{j=0}^{\infty} \left( \frac{-1}{p} \right)^j \right| = \sum_{j=0}^{k-1} (a_{k-j} - a) \left( \frac{-1}{p} \right)^j - a \sum_{j=k}^{\infty} \frac{1}{p^j}
\]

\[
= \sum_{j=0}^{N} (a_{k-j} - a) \left( \frac{-1}{p} \right)^j + \sum_{j=N+1}^{k-1} (a_{k-j} - a) \left( \frac{-1}{p} \right)^j - a \sum_{j=k}^{\infty} \frac{1}{p^j}
\]

\[
\leq \sum_{j=0}^{N} |a_{k-j} - a| \left( \frac{-1}{p} \right)^j + \sum_{j=N+1}^{k-1} |a_{k-j} - a| \left( \frac{-1}{p} \right)^j + |a| \sum_{j=k}^{\infty} \frac{1}{p^j}
\]

\[
= \sum_{j=0}^{N} |a_{k-j} - a| \left( \frac{1}{p^j} \right) + \sum_{j=N+1}^{k-1} |a_{k-j} - a| \left( \frac{1}{p^j} \right) + |a| \sum_{j=N+1}^{\infty} \frac{1}{p^j}
\]

\[
\leq \sup_{i \geq k-N} |a_i - a| \sum_{j=0}^{\infty} \frac{1}{p^j} + \sup_{i \geq 1} |a_i - a| \sum_{j=N+1}^{\infty} \frac{1}{p^j} + |a| \sum_{j=N+1}^{\infty} \frac{1}{p^j}
\]

\[
= \left( \frac{p}{p-1} \right) \sup_{i \geq k-N} |a_i - a| + (|a| + \sup_{i \geq 1} |a_i - a|) \frac{1/p^{N+1}}{1-1/p},
\]

where the first term in the last expression tends to 0 as \( k \to \infty \). Since \( N \in \mathbb{N} \) was arbitrary, and \( N \to \infty \) causes the second term in the last expression to tend to 0, we get that

\[
\lim_{k \to \infty} S_k = a \sum_{j=0}^{\infty} \left( \frac{-1}{p} \right)^j = a \frac{p}{p+1},
\]

and thus

\[
\lim_{k \to \infty} y_k = \lim_{k \to \infty} e^{z_k} = e^{\lim_{k \to \infty} z_k} = e^{\lim_{k \to \infty} S_k} = e^{a \rho (c)} = \left( e^{\log (c)} \right)^{\rho (c)} = p^{\rho (c)} = \left( \frac{1}{2 \rho (c)} \right)^{p/(p+1)}.
\]

Since \( p/(p+1) = p/p^2 = 1/p \) and \( y_k = \frac{|e_{k+1}|}{|e_k|^p} \), we get that the order of the secant method is \( \alpha = p = 1 + \sqrt{5}/2 \), the golden ratio, and the asymptotic error constant is

\[
\lambda = \lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^p} = \frac{f''(x^*)}{2f'(x^*)} \left( \frac{1}{p} \right). \]

Finally, in the case \( f''(x^*) = 0 \), we need to impose more conditions on \( f \) to find the order of convergence of the secant method here. The constants \( c_k \) don’t need to approach 0 monotonically (unless more regularity is assumed...), which may cause occasional blow-ups in \( \lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^p} \), thwarting an attempt at getting the exact order.

It is reassuring to note, however, that if \( f''(x^*) = 0 \), the secant method converges to the root faster than any other method (with the same initial iterates) that converges of order \{the golden ratio\}, in the sense that \( e_k/\hat{e}_k \to 0 \) if the errors \( \hat{e}_k \) represent those associated with the other method (since the latter method would have to satisfy \( |\hat{e}_{k+1}| = \hat{\lambda}_k |\hat{e}_k|^p = \hat{\lambda}_k |\hat{e}_k| |\hat{e}_k|^{p-1} = \hat{\lambda}_k |\hat{e}_k| |\hat{e}_k|^{p-2} = \hat{\lambda}_k |\hat{e}_k|^p |\hat{e}_{k-1}|, \) where \( \hat{\lambda}_k \to \hat{\lambda} > 0 \), and by comparing this to the analogous equation involving the \( e_k \)'s, and then using a limiting argument).
7 General fixed-point iteration

Suppose we are trying to find the fixed point of a continuous function \( g : \mathbb{R} \to \mathbb{R} \); i.e., we are trying to find an \( x^* \) such that \( x^* = g(x^*) \). One way to accomplish this is via fixed-point iteration. The method begins with an initial guess \( x_0 \) of the fixed point, and then further iterations are given by \( x_{k+1} = g(x_k) \). This may be visualized by alternating travelling horizontally from the curve to \( y = x \), and then travelling vertically back to the curve, and repeating this process. The arrows in the following graph depict this movement. The initial iterates whose paths are shown are \( x_0 = -.01, .01, 2.3, \) and \( 4 \). This is known as cobwebbing.

Suppose \( g \) maps some particular interval \([a, b]\) into itself. Then \( g \) must have a root there, by the IVT: set \( h : \mathbb{R} \to \mathbb{R} \), \( h(x) = g(x) - x \). \( h(a) = g(a) - a \geq 0 \) and \( h(b) = g(b) - b \leq 0 \), so the IVT guarantees a \( c \in [a, b] \) with \( 0 = h(c) = g(c) - c \). Rearranging this shows that \( c \) is a fixed point for \( g \). This works even if \( g \) is only defined on \([a, b]\), of course.

Suppose \( g \) is Lipschitz with Lipschitz constant \( \lambda < 1 \); i.e., there is a constant \( \lambda \in \mathbb{R} \), \( 0 < \lambda < 1 \) with \( |g(x) - g(y)| \leq \lambda |x - y| \) for all \( x, y \in \mathbb{R} \). Then there cannot be more than one fixed point for \( g \). For if there were two distinct fixed points \( x \) and \( y \), then \( x = g(x) \) and \( y = g(y) \), so we would get \( 0 < |x - y| = |g(x) - g(y)| \leq \lambda |x - y| < |x - y| \), a contradiction. So any fixed point must be unique. Let’s try fixed-point iteration on \( g \). If \( m > n \), we have the following estimate:

\[
|x_m - x_n| \leq |x_m - x_{m-1}| + |x_{m-1} - x_{m-2}| + \cdots + |x_{n+2} - x_{n+1}| + |x_{n+1} - x_n| \\
= |g(x_{m-1}) - g(x_{m-2})| + |x_{m-1} - x_{m-2}| + \cdots + |x_{n+2} - x_{n+1}| + |x_{n+1} - x_n| \\
\leq (1 + \lambda)|x_{m-1} - x_{m-2}| + |x_{m-2} - x_{m-3}| + \cdots + |x_{n+2} - x_{n+1}| + |x_{n+1} - x_n| \\
\leq (1 + \lambda + \lambda^2)|x_{m-2} - x_{m-3}| + |x_{m-3} - x_{m-4}| + \cdots + |x_{n+2} - x_{n+1}| + |x_{n+1} - x_n| \\
\leq \cdots \leq (1 + \lambda + \lambda^2 + \cdots + \lambda^{m-n-1})|x_{n+1} - x_n| \\
= \frac{1 - \lambda^{m-n}}{1 - \lambda}|x_{n+1} - x_n| \leq \frac{1 - \lambda^{m-n}}{1 - \lambda}|x_n - x_{n-1}| \leq \cdots \leq \frac{1 - \lambda^{m-n}}{1 - \lambda}\lambda^n|x_1 - x_0| \\
\leq \frac{\lambda^n}{1 - \lambda}|x_1 - x_0|.
\]
This implies the sequence \( \{x_n\} \) is Cauchy, and hence converges (in the complete metric space \( \mathbb{R} \)) to some \( x^* \in \mathbb{R} \). Furthermore, since Lipschitz implies continuous, we have 
\[
g(x^*) = g(\lim x_n) = \lim g(x_n) = \lim x_{n+1} = x^*, \]
showing that indeed \( x^* \) is a fixed point of \( g \). So we’ve proven that in the Lipschitz case with Lipschitz constant \( \lambda < 1 \), \( g \) admits a unique fixed point, and it is the limit of fixed-point iteration. Furthermore, we have a nice estimate of the error between our iterates and the fixed point, namely, 
\[
|x^* - x_n| = \lim_{m \to \infty} |x_m - x_n| = \lim_{m \to \infty} |x_m - x_n| = \limsup_{m \to \infty} |x_m - x_n| \leq \frac{\lambda^n}{1 - \lambda}|x_1 - x_0|.
\]
This is especially nice for a computer algorithm, since we know exactly how many iterations to perform to get within any desired error tolerance, just based on \( \lambda \) and the distance between the first two iterates. Note that if \( g' \) exists on \( \mathbb{R} \) and \( |g'| \leq \lambda < 1 \) on \( \mathbb{R} \), then the MVT implies that \( g \) is Lipschitz with Lipschitz constant \( 0 < \lambda < 1 \). Also, again, everything discussed here works in the setting \( g : [a, b] \to [a, b] \) Lipschitz with Lipschitz constant \( 0 < \lambda < 1 \).

Suppose \( g \) has \( n \geq 1 \) continuous derivatives in a neighborhood of the fixed point \( x^* \) such that \( g'(x^*) = g''(x^*) = \cdots = g^{(n-1)}(x^*) = 0 \), but \( g^{(n)}(x^*) \neq 0 \). If we attempt to perform fixed-point iteration on \( g \), then Taylor expansion yields
\[
e_{k+1} = x_{k+1} - x^* \\
= g(x_k) - g(x^*) \\
= g'(x^*)(x_k - x^*) + \frac{g''(x^*)}{2}(x_k - x^*)^2 + \cdots + \frac{g^{(n-1)}(x^*)}{(n-1)!}(x_k - x^*)^{n-1} + \frac{g^{(n)}(\xi)}{n!}(x_k - x^*)^n \\
= \frac{g^{(n)}(\xi)}{n!}(x_k - x^*)^n \\
= \frac{g^{(n)}(\xi)}{n!}(e_k)^n,
\]
demonstrating that if we start close enough to the fixed point \( x^* \), the method of fixed-point iteration converges of order \( \alpha = n \), with asymptotic error constant \( \lambda = \frac{|g^{(n)}(x^*)|}{n!} \). Note that if \( \alpha = 1 \), then we need \( |\lambda| < 1 \) to guarantee convergence. The condition on \( g \)'s derivatives corresponds, geometrically, to how flat the graph of \( g \) is. We can tell from looking at the cobwebbing example above that the flatter \( g \) is near \( x^* \) (i.e. the more of \( g \)'s derivatives that are 0 at \( x^* \)), the faster the algorithm will converge to the fixed point \( x^* \). Also, even if we don’t know what \( x^* \) is, we can often still tell how many of \( g \)'s derivatives will be 0 at the root.

Fixed-point problems and root-finding problems are very similar, and can always be easily transformed into one another. The fixed-point problem \( x = g(x) \) can be changed into the root-finding problem \( f(x) = \theta(x)(g(x) - x) = 0 \) for any zero-free function \( \theta \), and \( (x^* \) is a root of \( f \)) \iff \( x^* \) is a fixed point of \( g \). The root-finding problem \( f(x) = 0 \) can be changed into the fixed-point problem \( g(x) = x + \phi(x)f(x) \) for any zero-free function \( \phi \), and again \( (x^* \) is a fixed point of \( g \)) \iff \( x^* \) is a root of \( f \). Another common tactic to transform a root-finding problem to a fixed-point problem is to change one or more of the \( x \)'s to \( x_{k+1} \)'s, and change all the other \( x \)'s to \( x_k \)'s; then isolate \( x_{k+1} \); and finally, consider the RHS to be \( g(x_k) \). The goal is to form a \( g \) that is as flat as possible near the fixed point.
8 Accelerating convergence

Suppose \((p_n)\) converges linearly to \(p\) with asymptotic error constant \(0 < \lambda < 1\). So \(\lim_{n \to \infty} \frac{p_{n+1} - p}{p_n - p} = \lambda\). Note that this condition implies that eventually, no \(p_n\) is equal to \(p\).

If the iterates are obtained via fixed-point iteration, we would’ve stopped the algorithm from producing further iterates once the fixed point was obtained anyways, so this is a reasonable assumption. Put \(h: \mathbb{N} \to \mathbb{R}, h(n) := \frac{p_{n+1} - p}{p_n - p} - \lambda\). Then \(\lim_{n \to \infty} h(n) = 0\). So we can write:

\[
\frac{p_{n+1} - p}{p_n - p} - h(n) = \lambda = \frac{p_{n+2} - p}{p_{n+1} - p} - h(n + 1),
\]

which implies that

\[
(p_{n+1} - p)(p_{n+1} - p) + H(n)(p_n - p)(p_{n+1} - p) = (p_{n+2} - p)(p_n - p),
\]

where \(H(n) = h(n + 1) - h(n)\). Then

\[
p + H(n) \frac{(p_n - p)(p_{n+1} - p)}{p_{n+2} - 2p_{n+1} + p_n} = \frac{p_{n+2}p_n - p_{n+1}^2}{p_{n+2} - 2p_{n+1} + p_n}.
\]

Call the expression on the right \(\hat{p}_n\), so that

\[
\hat{p}_n = \frac{p_{n+2}p_n - p_{n+1}^2}{p_{n+2} - 2p_{n+1} + p_n} = p_n - \frac{p_n - (p_{n+1} - p)^2}{p_{n+2} - 2p_{n+1} + p_n}.
\]

Note that

\[
p_{n+2} - 2p_{n+1} + p_n = (p_{n+2} - p) - 2(p_{n+1} - p) + (p_n - p) = (p_{n+1} - p) \left( \frac{p_{n+2} - p}{p_{n+1} - p} - 2 + \frac{p_n - p}{p_{n+1} - p} \right) = (p_{n+1} - p) \left( \lambda + h(n + 1) - 2 + \frac{1}{\lambda + h(n)} \right),
\]

which means that \(\frac{p_{n+2} - 2p_{n+1} + p_n}{p_{n+1} - p} \to \lambda - 2 + \frac{1}{\lambda} = \frac{\lambda^2 - 2\lambda + 1}{\lambda} = \frac{(\lambda - 1)^2}{\lambda} \neq 0\) as \(n \to \infty\). This retroactively justifies the division by this expression in the formula for \(\hat{p}_n\).

Since \(\frac{\hat{p}_n - p}{p_n - p} = H(n) \frac{p_{n+1} - p}{p_{n+2} - 2p_{n+1} + p_n} \to (0) \left( \frac{\lambda}{(\lambda - 1)^2} \right) = 0\) as \(n \to \infty\). This method of converting the original sequence elements \(p_n\) to the modified sequence elements \(\hat{p}_n\) is known as Aitken’s method, and we’ve just shown that it “speeds up” the convergence to the (common) limit \(p\). In fact, this even works if \(\lambda = 0\), by a slight modification of this argument. Note that “accelerated convergence” alone doesn’t actually guarantee that the new iterates even converge of any order \(\alpha > 0\); the new iterates \(\hat{p}_n\) might have errors \(\hat{e}_n\) that shrink to 0 far faster than the corresponding errors associated with the original sequence, but still stagnate/grow occasionally (e.g. \(|\hat{e}_{n+1}| = |\hat{e}_n|^{1/n}\) for infinitely many \(n\)).

Steffensen’s method, a method of accelerating convergence based on Aitken’s, is outlined in the textbook.
9 Computer arithmetic

read lecture notes/textbook!

10 MATLAB demo

- explanation of the sample codes given for assignments
- sprintf (inherited from C/C++. Similar: fprintf)
- disp
- break vs. return
- eval
- creating functions, e.g.

\[ f = @(x) x.^2 - 3; \]