Solving a Nonlinear Equation

Reading: Cheney & Kincaid Chapter 3

Problem: Given $f(x)$, find a root (solution) of $f(x) = 0$

- If f is a polynomial with degree 4 or less, formulas for roots exist.
- If f is a polynomial with degree larger than 4, **no formula exists** (proved in the late 19th century).
- If f is a general nonlinear function, no formula exists

So we must be satisfied by a method which only computes **approximate** roots.

Iterative Methods

Since no formula exists for roots of $f(x) = 0$, **iterative methods** will be used to compute **approximate roots.**

Iterative methods construct a sequence of numbers $x_1, x_2, \ldots, x_n, \ldots$ that converge to a root of $f(x) = 0$.

3 major issues with implementation of iterative methods:

- Where to start the iteration?
- Does the iteration converge, and how fast?
- When to terminate the iteration?

3 iterative methods will be introduced in this course:

- Bisection method
- Newton's method
- Secant method

The Bisection Method

Fact: If $f(x)$ is continuous on $[a, b]$ and $f(a) * f(b) < 0$, then there exists r such that $f(r) = 0$.

Idea: The fact can be used to produce a sequence of ever-smaller intervals that each brackets a root of $f(x) = 0$: Let $c = (a + b)/2$ (midpoint of $[a, b]$). Compute $f(c)$.

- If $f(c) = 0$, c is a root.
-
  - If $f(a)f(c) < 0$, a root exists in $[a, c]$.

– If $f(c)f(b) < 0$, a root exists in $[c, b]$.

In either case, the interval is half as long as the initial interval. The halving process can continue until the current interval is shorter than a given tolerance δ.

The algorithm

Algorithm. Given $f(x)$, $[a, b]$ with $f(a)f(b) < 0$, and the tolerance δ

```
c ← (a + b)/2
error_bound ← |b − a|/2
while error_bound > δ
      if f(c) = 0, then c is a root, stop.
      else
           if f(a)f(c) < 0, then
              b ← c
           else
              a ← c
           end
      end
      c ← (a + b)/2
      error_bound ← error_bound/2
end
root ← c
```

Note: When implementing this algorithm, avoid recomputation of values of function, and use $\text{sign}(f(a))\text{sign}(f(c)) < 0$ instead of $f(a)f(c) < 0$ to avoid overflow and underflow.

Matlab code

```
function root = bisection(fname,a,b,delta,display)
% The bisection method.
%
%input: fname is a string that names the function f(x)
%       a and b define an interval [a,b]
%       delta is a tolerance
%       display = 1 if step-by-step display is desired,
%               = 0 otherwise
%output: root is the computed root of f(x)=0
%
fa = feval(fname,a);
fb = feval(fname,b);
if sign(fa)*sign(fb) > 0
   disp('function has the same sign at a and b')
   return
end
if fa == 0, root = a; return; end
if fb == 0, root = b; return; end
```

```
c = (a+b)/2;
fc = feval(fname,c);
e_bound = abs(b-a)/2;
if display,
   disp(' ');
   disp('    a       b       c      f(c)     error_bound');
   disp(' ');
   disp([a b c fc e_bound])
end
while e_bound > delta
   if fc == 0,
      root = c;
      return
   end
   if sign(fa)*sign(fc) < 0  % a root exists in [a,c].
      b  = c;
      fb = fc;
   else                      % a root exists in [c,b].
      a  = c;
      fa = fc;
   end
   c = (a+b)/2;
   fc = feval(fname,c);
   e_bound = e_bound/2;
   if display, disp([a b c fc e_bound]), end
end
root = c;
```

Convergence and efficiency

Suppose the initial interval is $[a, b] \equiv [a_0, b_0]$ and r is a root. At the beginning ($n = 0$),

$$c_0 = \frac{1}{2}(a_0 + b_0), \quad |r - c_0| \le \frac{1}{2}|b_0 - a_0|.$$

After n steps, we get interval $[a_n, b_n]$, $c_n = \frac{1}{2}(a_n + b_n)$,

$$|b_n - a_n| = \frac{1}{2}|b_{n-1} - a_{n-1}|, \quad |r - c_n| \le \frac{1}{2}|b_n - a_n|.$$

Therefore we have

$$|r - c_n| \le \frac{1}{2^2}|b_{n-1} - a_{n-1}| = \cdots = \frac{1}{2^{n+1}}|b - a|,$$

$$\lim_{n\to\infty} c_n = r.$$

Q. How many steps are required to ensure $|r - c_n| \le \delta$ for a general continuous function?
To ensure $|r - c_n| \le \delta$, we require $\frac{1}{2^{n+1}}|b - a| \le \delta$. From this we obtain

$$n \ge \log_2(|b - a|/\delta) - 1.$$

So $\lceil \log_2(|b-a|/\delta) - 1 \rceil$ steps are needed.

Def. Linear convergence: A sequence $\{x_n\}$ is said to have linear convergence to x if

$$\lim_{n\to\infty} \frac{|x_{n+1} - x|}{|x_n - x|} = c, \quad 0 < c < 1.$$

Obviously the right hand side of $|r - c_n| \leq \frac{1}{2^{n+1}}|b-a|$ has linear convergence to zero. Although $\{c_n\}$ does not have linear convergence to r (check this), we still say the order of convergence of the bisection method is linear.

Note: The bisection method uses **sign** information only. Given an interval in which a root lies, it maintains a guaranteed interval, but is slow to converge. If we use more information, such as values or derivatives of f, we can get faster convergence.

Newton's Method

Idea: Given a point x_0. Taylor series expansion about x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(z)}{2}(x - x_0)^2.$$

We can use $l(x) \equiv f(x_0) + f'(x_0)(x - x_0)$ as an approximation to $f(x)$. In order to solve $f(x) = 0$, we solve $l(x) = 0$, which has the solution

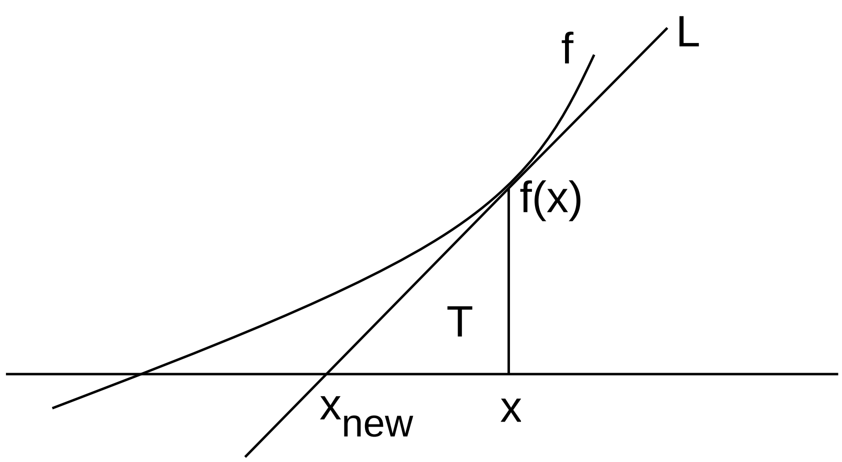
$$x_1 = x_0 - f(x_0)/f'(x_0).$$

So x_1 can be regarded as an approximate root of $f(x) = 0$. If this x_1 is not a good approximate root, we continue this iteration. In general we have the Newton iteration:

$$x_{n+1} = x_n - f(x_n)/f'(x_n), \quad n = 0, 1, \ldots.$$

Here we assume $f(x)$ is differentiable and $f'(x_n) \neq 0$.

Understand Newton's method geometrically



The line L is tangent to f at $(x, f(x))$, and so has slope $f'(x)$.
The slope of the line L is $f(x)/(x - x_{\text{new}})$, so:

$$f'(x) = \frac{f(x)}{x - x_{\text{new}}},$$

and consequently

$$x_{\text{new}} = x - \frac{f(x)}{f'(x)}.$$

This is just the Newton iteration.

The algorithm

Stopping criteria

- $|x_{n+1} - x_n| \leq xtol$, or
- $|f(x_{n+1})| \leq ftol$, or
- The maximum number of iteration reached.

Algorithm. Given f, f', x (initial point), $xtol$, $ftol$, n_{max} (the maximum number of iterations):

for $n = 1 : n_{max}$
  $d \leftarrow f(x)/f'(x)$
  $x \leftarrow x - d$
  if $|d| \leq xtol$ or $|f(x)| \leq ftol$, then
    $root \leftarrow x$
    stop
  end
end
$root \leftarrow x$

Matlab code

```
function root=newton(fname,fdname,x,xtol,ftol,n_max,display)
% Newton's method.
% input:
%   fname   string that names the function f(x).
%   fdname  string that names the derivative f'(x).
%   x       the initial point
%   xtol and ftol  termination tolerances
%   nmax    the maximum number of iteration
%   display = 1 if step-by-step display is desired,
%           = 0 otherwise
% output: root is the computed root of f(x)=0
%
```

```
n = 0;
fx = feval(fname,x);
if display,
   disp('   n           x            f(x)')
   disp('------------------------------------')
   disp(sprintf('%4d %23.15e %23.15e', n, x, fx))
end
if abs(fx) <= xtol
   root = x;
   return
end
for n = 1:nmax
    fdx = feval(fdname,x);
    d = fx/fdx;
    x = x - d;
    fx = feval(fname,x);
    if display,
       disp(sprintf('%4d %23.15e %23.15e',n,x,fx))
    end
    if abs(d) <= xtol | abs(fx) <= ftol
       root = x;
       return
    end
end
```

The function $f(x)$ and its derivative $f'(x)$ are implemented by Matlab, for example, for $f(x) = x^2 - 2$,

```
% M-file f.m
function y = f(x)
y = x^2 -2;
```

```
% M-file fd.m
function y = fd(x)
y = 2*x;
```

Example: $f(x) = x^2 - 2$

```
>> root=newton('f','fd',2,1.e-12,1.e-12,20,1)
   n             x                   f(x)
------------------------------------------------
   0 2.000000000000000e+00 2.000000000000000e+00
   1 1.500000000000000e+00 2.500000000000000e-01
   2 1.416666666666667e+00 6.944444444444642e-03
   3 1.414215686274510e+00 6.007304882871267e-06
   4 1.414213562374690e+00 4.510614104447086e-12
   5 1.414213562373095e+00 4.440892098500626e-16
root =
     1.414213562373095e+00
```

It gives **double precision** accuracy in **only 5 steps**!

In steps 2, 3, 4: $|f(x)| \approx 10^{-3}$, $|f(x)| \approx 10^{-6}$, $|f(x)| \approx 10^{-12}$, respectively. We say $f(x)$ converges to 0 with **quadratic convergence**: once $|f(x)|$ is small, it is roughly **squared**, and thus **much smaller**, at the next step.

In steps 2, 3, 4: x is accurate to about **3 decimal digits**, **6 decimal digits**, **12 decimal digits**, respectively. The number of accurate digits of x is approximately doubled at each step. We say x converges to the root with **quadratic convergence**.

Failure of Newton's method

Newton's method does not always works well. It may not converge.

- If $f'(x_n) = 0$ the method is not defined.
- If $f'(x_n) \approx 0$ then there may be difficulties. The new approximate x_{n+1} may be a much worse approximation to the solution than x_n is.

Some examples will be given in class.

Convergence analysis

Def. Quadratic convergence (QC). A sequence $\{x_n\}$ is said to have *quadratic convergence* to x if

$$\lim_{n\to\infty} \frac{|x_{n+1} - x|}{|x_n - x|^2} = c,$$

where c is some finite non-zero constant.

Newton iteration:

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

Suppose x_n converges to a root r of $f(x) = 0$. In the following we show usually it has quadratic convergence.

The Taylor series expansion about x_n is

$$f(r) = f(x_n) + (r - x_n)f'(x_n) + \frac{(r - x_n)^2}{2} f''(z_n)$$

where z_n lies between x_n and r. Dividing both sides of the above equality by $f'(x_n)$ and using $f(r) = 0$ gives

$$0 = \frac{f(x_n)}{f'(x_n)} + (r - x_n) + (r - x_n)^2 \frac{f''(z_n)}{2f'(x_n)}.$$

Here $\frac{f(x_n)}{f'(x_n)} = x_n - x_{n+1}$, so

$$r - x_{n+1} = c_n (r - x_n)^2, \quad c_n \equiv -\frac{f''(z_n)}{2f'(x_n)}.$$

Writing the **error** $e_n = r - x_n$, we see

$$e_{n+1} = c_n (e_n)^2.$$

Since $x_n \to r$, and z_n is between x_n and r, we see $z_n \to r$ and so

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^2} = \lim_{n\to\infty} |c_n| = \frac{|f''(r)|}{|2f'(r)|} \equiv c,$$

where we assume that $f'(r) \neq 0$.

Therefore, the convergence rate of Newton method is **usually quadratic**. At the $(n+1)$-th step, the error is equal to c_n times the **square** of the error at the n-th step.

Note: We can also show that $f(x_n)$ usually converges to 0 **quadratically**:

$$\lim_{n\to\infty} |f(x_{n+1})|/|f(x_n)|^2 = c'$$

Q. Under which condition does x_n converge r?
It can be shown that if $f''(x)$ is continuous in a neighborhood of r, $f'(r) \neq 0$, and if the initial point x_0 is close to r enough, then $x_n \to r$. (For a rigorous proof, see C&K, pp.129-130.)

Notes:

- If $f''(r) = 0$ but $f'(r) \neq 0$, then

  $$\lim_{n\to\infty} |e_{n+1}/e_n^2| = 0.$$

  NM converges faster than quadratic convergence.
  e.g., $f(x) = \sin(x)$, $r = \pi$.
  Try `newton('f','fd',4,1.e-12,1.e-12,20,1)`

- If $f'(r) = 0$, then we can show NM has linear convergence rate.
  e.g., $f(x) = (x-1)^2$, $r = 1$.
  Try `newton('f','fd',1.5,1.e-12,1.e-12,20,1)`
  For this specific function, try to show NM has linear convergence rate.

- If the initial point is not close to the root r, NM may not converge.
  e.g., $f(x) = \arctan(x)$, $r = 0$, $x_0 = 1.5$.
  Try `newton('f','fd',1.5,1.e-12,1.e-12,20,1)`

  In practice, one usually has some rough idea about where a root he/she seeks. One can also draw the graph of $f(x)$ to get some idea where the roots are.

The Secant Method

Idea: One problem with Newton iteration ($x_{n+1} = x_n - f(x_n)/f'(x_n)$) is it requires software computing the derivative $f'(x)$ — in many instances, difficult or impossible to compute. One idea to get around the problem is to use **divided difference** $\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}$ to replace $f'(x_n)$. Then we get the secant iteration:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

This formula can be understood geometrically:
Draw a secant line which connects two points $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$ on the graph of

$y = f(x)$. The cross point of the secant line and the x-axis is exactly the x_{n+1} defined by the secant iteration formula.

Algorithm. Given f, x_0, x_1 (two initial points), $xtol$, $ftol$, n_{max}

for $n = 1 : n_{max}$
    $d \leftarrow \frac{x_1 - x_0}{fx_1 - fx_0} fx_1$
    $x_0 \leftarrow x_1$
    $fx_0 \leftarrow fx_1$
    $x_1 \leftarrow x_1 - d$
    $fx_1 \leftarrow f(x_1)$
    if $|d| \leq xtol$ or $|f(x)| \leq ftol$, then
        $root \leftarrow x_1$
        return
    end
end
$root \leftarrow x_1$

Convergence result

If x_n converges to a root r of $f(x) = 0$, we can show (somewhat difficult) that

$$\lim_{k\to\infty} \frac{|x_{n+1} - r|}{|x_n - r|^p} = c$$

where $p = (1 + \sqrt{5})/2 \approx 1.618$. So the secant method converges **super-linearly.**
The secant method may not converge for much the same reason that Newton's method may not converge. For example, the method is not defined if $f(x_n) = f(x_{n-1})$.

Comparison of the Three Methods

- **The Bisection Method (BM)**:
  - Advantages: simple, robust (guaranteed to converge), applicable to nonsmooth functions.
  - Disadvantages: generally slower than NM and SM with linear convergence.
- **Newton's Method (NM)**:
  - Advantages: generally faster than BM and SM with quadratic convergence.
  - Disadvantages: needs to compute f', may not converge.
- **The Secant Method (SM)**:
  - Advantages: generally faster than BM with super-linear convergence, no need to compute f'.
  - Disadvantages: slower than NM, may not converge.

<u>Two MATLAB Commands</u>

There are two Matlab built-in functions:

- `roots` for finding all roots of a polynomial.
- `fzero` for finding a root of a general function.

Check Matlab to see how to use these two functions.