Numerical Analysis, lecture 5:

Finding roots
(textbook sections 4.1–3)

- introducing the problem
- bisection method
- Newton-Raphson method
- secant method
- fixed-point iteration method
We need methods for solving nonlinear equations (p. 64-65)

Problem: Given \( f : \mathbb{R} \to \mathbb{R} \), find \( x^* \) such that \( f(x^*) = 0 \).

Numerical methods are used when

- there is no formula for root,
- the formula is too complex,
- \( f \) is a “black box”
Roots can be simple or multiple \((\text{p. 66})\)

\(x^*\) is a root of \(f\) having multiplicity \(q\) if

\[
f(x) = (x - x^*)^q g(x) \text{ with } g(x^*) \neq 0
\]

\[
f(x^*) = f'(x^*) = \cdots = f^{(q-1)}(x^*) = 0 \text{ and } f^{(q)}(x^*) \neq 0
\]
First: get an estimate of the root location (p. 66-67)

use theory

All roots of \( x - \cos x = 0 \) lie in the interval \([-1,1]\)

\[
| x | = | \cos x | \leq 1
\]

use graphics

3 rules
1. graph the function
2. make a graph of the function
3. make sure that you have made a graph of the function

difficult cases:

0, 1 or 2 roots?

many roots

pole
Bisection traps a root in a shrinking interval (p. 67-68)

Bracketing-interval theorem

If \( f \) is continuous on \([a,b]\) and \( f(a) \cdot f(b) < 0 \) then \( f \) has at least one zero in \((a,b)\).

Bisection method

Given a bracketing interval \([a,b]\), compute \( x = \frac{a + b}{2} \) & \( \text{sign}(f(x)) \);
repeat using \([a,x]\) or \([x,b]\) as new bracketing interval.

```matlab
function x=bisection(f,a,b,tol)
sfb = sign(f(b));
width = b-a;
    disp(' a            b         sfx')
while width > tol
    width = width/2;
    x = a + width;
    sfx = sign(f(x));
    disp(sprintf('%0.8f   %0.8f   %2.0f', [a b sfx]))
    if sfx == 0, a = x; b = x; return
    elseif sfx == sfb, b = x;
    else, a = x; end
end
```

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{sfx} \\
0.70000000 & 0.80000000 & 1 \\
0.70000000 & 0.75000000 & -1 \\
0.72500000 & 0.75000000 & -1 \\
0.73750000 & 0.75000000 & 1 \\
0.73750000 & 0.74375000 & 1 \\
0.73750000 & 0.74062500 & -1 \\
0.73906250 & 0.74062500 & 1 \\
\end{array}
\]
Bisection is slow but dependable (p. 68)

Advantages

• guaranteed convergence
• predictable convergence rate
• rigorous error bound

Disadvantages

• may converge to a pole
• needs bracketing interval
• slow
Newton-Raphson method uses the tangent

**Iteration formula**

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

```plaintext
function x=newtonraphson(f,df,x,nk)
disp('k     x_k              f(x_k)    f''(x_k)      dx')
for k = 0:nk
    dx = df(x)(x);
    disp(sprintf('%d   %0.12f   %9.2e   %1.5f   %15.12f',[k,x,f(x),df(x),dx]))
    x = x - dx;
end

>> f = @(x) x-cos(x); df = @(x) 1+sin(x);
>> newtonraphson(f,df,0.7,3);
```

<table>
<thead>
<tr>
<th>k</th>
<th>x_k</th>
<th>f(x_k)</th>
<th>f''(x_k)</th>
<th>dx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.700000000000</td>
<td>-6.48e-02</td>
<td>1.64422</td>
<td>-0.039436497848</td>
</tr>
<tr>
<td>1</td>
<td>0.739436497848</td>
<td>5.88e-04</td>
<td>1.67387</td>
<td>0.000351337383</td>
</tr>
<tr>
<td>2</td>
<td>0.739085160465</td>
<td>4.56e-08</td>
<td>1.67361</td>
<td>0.000000027250</td>
</tr>
<tr>
<td>3</td>
<td>0.739085133215</td>
<td>2.22e-16</td>
<td>1.67361</td>
<td>0.000000000000</td>
</tr>
</tbody>
</table>
Newton-Raphson is fast  (p. 70)

Advantages
- quadratic convergence near simple root
- linear convergence near multiple root

Disadvantages
- iterates may diverge
- requires derivative
- no practical & rigorous error bound

Exercise
Find the first positive root of \( x = \tan x \)
Secant method is derivative-free (p. 72-73)

**Iteration formula**

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

```
function xx = secant(f,xx,nk)
disp('k   x_k           f(x_k)')
ff = [f(xx(1)), f(xx(2))];
h = 10*sqrt(eps);
for k = 0:nk
    disp(sprintf('%d %17.14f %14.5e',...          
        [k,xx(1),ff(1)]))
    if abs(diff(xx)) > h
        df = diff(ff)/diff(xx);
    else
        df = (f(xx(2)+h)-ff(2))/h;
    end
    xx = [xx(2), xx(2)-ff(2)/df];  % update xx
    ff = [ff(2), f(xx(2))];        % update ff
end
```

```
>> f = @(x) x-cos(x);
>> secant(f,[0.7 0.8],6);

<table>
<thead>
<tr>
<th>k</th>
<th>x_k</th>
<th>f(x_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.70000000000000</td>
<td>-6.48422e-02</td>
</tr>
<tr>
<td>1</td>
<td>0.80000000000000</td>
<td>1.03293e-01</td>
</tr>
<tr>
<td>2</td>
<td>0.73856544025090</td>
<td>-8.69665e-04</td>
</tr>
<tr>
<td>3</td>
<td>0.73907836214467</td>
<td>-1.13321e-05</td>
</tr>
<tr>
<td>4</td>
<td>0.73908513321516</td>
<td>1.30073e-09</td>
</tr>
<tr>
<td>5</td>
<td>0.73908513321516</td>
<td>-1.77636e-15</td>
</tr>
<tr>
<td>6</td>
<td>0.73908513321516</td>
<td>0.00000e+00</td>
</tr>
</tbody>
</table>
```
Secant method is also fast  (p. 73)

Advantages
- better-than-linear convergence near simple root
- linear convergence near multiple root
- no derivative needed

Disadvantages
- iterates may diverge
- no practical & rigorous error bound
Without bracketing, an iteration can jump far away \( \text{(p. 82)} \)

**Example**

```matlab
>> f = @(x) 1/x - 1;
>> secant(f,[0.5,10],4);
```

<table>
<thead>
<tr>
<th>k</th>
<th>x_k</th>
<th>f(x_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>-9.000000e-01</td>
</tr>
<tr>
<td>2</td>
<td>5.5</td>
<td>-8.18182e-01</td>
</tr>
<tr>
<td>3</td>
<td>-39.5</td>
<td>-1.02532e+00</td>
</tr>
<tr>
<td>4</td>
<td>183.25</td>
<td>-9.94543e-01</td>
</tr>
</tbody>
</table>

```matlab
>> df = @(x) -1/x^2;
>> newtonraphson(f,df,10,3);
```

<table>
<thead>
<tr>
<th>k</th>
<th>x_k</th>
<th>f(x_k)</th>
<th>f'(x_k)</th>
<th>dx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>-9.00e-01</td>
<td>-0.0100</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>-80</td>
<td>-1.01e+00</td>
<td>-0.0016</td>
<td>6480</td>
</tr>
<tr>
<td>2</td>
<td>-6560</td>
<td>-1.00e+00</td>
<td>-0.0000</td>
<td>4.30402e+07</td>
</tr>
<tr>
<td>3</td>
<td>-4.30467e+07</td>
<td>-1.00e+00</td>
<td>-0.0000</td>
<td>1.85302e+15</td>
</tr>
</tbody>
</table>
```
Illinois method is a derivative-free method with bracketing and fast convergence

False position (or: regula falsi) method combines secant with bracketing: it is slow

```
function x=illinois(f,a,b,tol)
    fa=f(a); fb=f(b);
    while abs(b-a)>tol
        step=fa*(a-b)/(fb-fa);
        x=a+step;
        fx=f(x);
        if sign(fx)~=sign(fb)
            a=b; fa=fb;
        else
            fa=fa/2;
        end
        b=x; fb=fx;
    end
```
Brent’s method combines bisection, secant and inverse quadratic interpolation (p. 83-84)

```matlab
>> f = @(x) 1./x - 1;
>> opts = optimset('display','iter','tolx',1e-10);
>> x = fzero(f,[0.5,10],opts)
```

<table>
<thead>
<tr>
<th>Func-count</th>
<th>x</th>
<th>f(x)</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>-0.9</td>
<td>initial</td>
</tr>
<tr>
<td>3</td>
<td>5.5</td>
<td>-0.818182</td>
<td>interpolation</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>-0.666667</td>
<td>bisection</td>
</tr>
<tr>
<td>5</td>
<td>1.75</td>
<td>-0.428571</td>
<td>bisection</td>
</tr>
<tr>
<td>6</td>
<td>1.125</td>
<td>-0.111111</td>
<td>bisection</td>
</tr>
<tr>
<td>7</td>
<td>0.953125</td>
<td>0.0491803</td>
<td>interpolation</td>
</tr>
<tr>
<td>8</td>
<td>1.00586</td>
<td>-0.00582524</td>
<td>interpolation</td>
</tr>
<tr>
<td>9</td>
<td>1.00027</td>
<td>-0.000274583</td>
<td>interpolation</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>7.54371e-08</td>
<td>interpolation</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>-2.07194e-11</td>
<td>interpolation</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>-2.07194e-11</td>
<td>interpolation</td>
</tr>
</tbody>
</table>

Zero found in the interval [0.5, 10]

x =

1.00000000002072
Root-finding can be treated as fixed-point-finding (p. 70-72)

Fixed point problem

Given \( \varphi : \mathbb{R} \rightarrow \mathbb{R} \), find \( x^* \) such that \( \varphi(x^*) = x^* \).

Fixed-point iteration

\[ x_{k+1} = \varphi(x_k) \]

Example (p. 71)
enter 0.7 on your calculator
press \texttt{cos} repeatedly

Example (p. 72)
instead, press \texttt{arccos} repeatedly
Newton-Raphson method is also a fixed point iteration (p. 70)

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

this insight will be the basis for the convergence analysis (next lecture)
what happened, what’s next

• first, localize the root
• bisection is dependable but slow
• Newton-Raphson & secant are fast if the initial value is good
• root-finding methods can be treated as fixed-point iterations

Next lecture: convergence, stopping criteria (§4.4-5)
Numerical Analysis, lecture 6:

Finding roots II
(textbook sections 4.4–5)

• convergence
• error estimate & achievable accuracy
• stopping criteria
In lecture 5 we learned about several root-finding algorithms

**bisection method**

Given a bracketing interval \([a,b]\), compute \(x = \frac{a + b}{2}\) & \(\text{sign}(f(x))\);

repeat using \([a,x]\) or \([x,b]\) as new bracketing interval.

**Newton-Raphson method**

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

**secant method**

\[ x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})} \] \(x_k - x_{k-1}\)
Fixed point iteration can be used to find roots (p. 70-72)

Fixed-point iteration

\[ x_{k+1} = \varphi(x_k) \]

for example:

\[ x - \cos x = 0 \quad \Rightarrow \quad \begin{cases} 
  x_{k+1} = \cos x_k \\
  \text{or} \\
  x_{k+1} = \cos^{-1} x_k \\
  \text{or} \\
  x_{k+1} = x_k - \frac{x_k - \cos x_k}{1 + \sin x_k} \\
  \text{or...} 
\end{cases} \]

(Newton-Raphson)
Convergence near $x^*$ depends on $\varphi'(x^*)$ (p. 71)

$$x_{k+1} = \varphi(x_k)$$

- $-1 < \varphi'(x^*) < 0$
- $\varphi'(x^*) < -1$
- $\varphi'(x^*) = 0$

**Exercise:** draw the “cobweb diagram” for $0 < \varphi'(x^*) < 1$ and $\varphi'(x^*) > 1$
Iteration converges if $|\varphi'| \leq m < 1$ 
in a neighbourhood of $x^*$  
(p. 74-75)

**Theorem** (p. 74)

Let $\phi(x^*) = x^*$ and $x_{k+1} = \phi(x_k)$ for $k = 0, 1, 2, \ldots$

If $\max_{|x-x^*| \leq \delta} |\varphi'(x)| \leq m < 1$ and $|x_0 - x^*| \leq \delta$ then

a) $|x_k - x^*| \leq \delta$ for $k = 1, 2, \ldots$

b) $x_k \to x^*$

c) $x^*$ is the only fixed point of $\phi$ in $x^* \pm \delta$

**proof**

a) $x_k - x^* = \phi(x_{k-1}) - \phi(x^*)$

$= (x_{k-1} - x^*)\varphi'(\xi)$

b) $|x_k - x^*| \leq |x_0 - x^*| \cdot m^k$

c) $x^{**} = \phi(x^{**}), x^{**} \neq x^*$

$\Rightarrow |x^{**} - x^*| = |(x^{**} - x^*)\varphi'(\xi)| \leq m|x^{**} - x^*|$

$\Rightarrow \nabla$

**Example:**

$\cos$ has a fixed point near 0.74, and $|\sin 0.74| \approx 0.67 < 1$,

so $x_{k+1} = \cos x_k$ with $x_0 \approx 0.74$ converges to this fixed point.
Another example of a fixed-point iteration

**Problem:** Prove that this iteration converges
(in the neighborhood of a positive fixed point)

\[ x_{k+1} = e^{-x_k} \]

**Solution:**

\[ \varphi(x) = e^{-x} \quad \Rightarrow \quad \varphi'(x) = -e^{-x} \quad \Rightarrow \quad |\varphi'(x)| < 1 \quad (x > 0) \]

\[
>> \quad x=1; \text{for } k=1:10, x=\text{exp}(-x); \text{disp}(x), \text{end}
\]

0.3679  
0.6922  
0.5005  
0.6062  
0.5454  
0.5796  
0.5601  
0.5711  
0.5649  
0.5684
Newton-Raphson converges if \( x_0 \approx x^* \) \( \text{ (p. 75) } \)

\[
\phi(x) = x - \frac{f(x)}{f'(x)} \implies \phi'(x) = \frac{f(x)f''(x)}{(f'(x))^2}
\]

**simple root** \( \phi'(x^*) = 0 \)

**multiple root**

\[
f(x) = (x - x^*)^q g(x), \quad g(x^*) \neq 0 \implies \lim_{x \to x^*} \phi'(x) = 1 - \frac{1}{q}
\]

**proof**

\[
f(x) = (x - x^*)^q g(x)
\]

\[
f'(x) = q(x - x^*)^{q-1} g(x) + (x - x^*)^q g'(x)
\]

\[
f''(x) = q(q-1)(x - x^*)^{q-2} g(x) + 2q(x - x^*)^{q-1} g'(x) + (x - x^*)^q g''(x)
\]

\[
\phi'(x) = \frac{g(x)}{\left[ qg(x) + (x - x^*) g'(x) \right]^2}
\]

the higher the multiplicity, the slower the convergence!
Newton-Raphson converges quadratically to a simple root \((p. 76-77)\)

**order-\(p\) convergence**

\[
\frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} \to C
\]

if \(\phi'(x^*) < 1 \& \phi'(x^*) \neq 0\), convergence is linear \((p = 1)\) with \(C = |\phi'(x^*)| < 1\)

Newton-Raphson has quadratic convergence \((p = 2)\) to simple root, \(C = \left| \frac{f''(x^*)}{2f'(x^*)} \right|\)

Proof: \(\phi(x_k) = \phi(x^* + (x_k - x^*)) = \phi(x^*) + (x_k - x^*) \cdot \phi'(x^*) + \frac{1}{2} (x_k - x^*)^2 \cdot \phi''(x^*) + \cdots\)

Secant method has superlinear convergence \((p \approx 1.618)\) to simple root

Bisection method is bounded-above by a linearly converging iteration with \(C = \frac{1}{2}\)
Newton-Raphson can compute square roots (p. 76-77)

\[ f(x) = x^2 - a \Rightarrow x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} = \frac{1}{2} \left( x_k + \frac{a}{x_k} \right) \]

\[ C = \left| \frac{f''(x^*)}{2f'(x^*)} \right| = \frac{1}{2x^*} \]

```
>> a=3; x(1)=2; for k=1:3, x(k+1)=(x(k)+a/x(k))/2; end

>> err = x-sqrt(3)

err =

0.26794919243112   0.01794919243112   0.00009204957398   0.00000000244585

>> err(2:4)./(err(1:3).^2)

ans =

0.250000000000000   0.28571428571479   0.28865978697703
```
Error estimates tell us how close $x_k$ is to $x^*$ (p. 78-79)

Error estimate for any method

$$|f - \tilde{f}| \leq \delta, |f'| \geq M \implies |x_k - x^*| \leq \frac{\tilde{f}(x_k)}{M}$$

**proof:**

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

$$|x_k - x^*| \leq \left| \frac{f(x_k) - \tilde{f}(x_k) + \tilde{f}(x_k)}{f'(\xi)} \right| \leq \frac{\delta + \tilde{f}(x_k)}{M}$$

**Example**

0.73909 is an approximation of a zero of $f(x) = x - \cos x$ that has 5 correct decimals because

$$|0.73909 - x^*| \leq \frac{\tilde{f}(0.73909)}{\tilde{f}'(0.73909)} = \frac{8.15 \times 10^{-6}}{1.67} \leq 0.5 \times 10^{-5}$$

(here $\delta$ is assumed to be negligible)
The attainable accuracy is the most that you should expect from any method (p. 79-80)

\[ |x_k - x^*| \leq \frac{\delta}{M} \]

If \( f'(x^*) \approx 0 \), the root is ill-conditioned
Multiple roots are ill-conditioned (p. 80)

attainable accuracy of root of multiplicity $q$

$$|f - \tilde{f}| \leq \delta, \left| f^{(q)} \right| \geq M_q$$

$$f(x) = (x - x^*)^q g(x) \implies x_k - x^* \leq \left( \frac{q! \delta}{M_q} \right)^{1/q}$$

proof:

$$f(x_k) = f(x^*) + (x_k - x^*) f'(x^*) + \cdots + \frac{1}{q!} (x - x^*)^q f^{(q)}(\xi)$$

Example

$$p(x) = (x - 1)^7 = x^7 - 7x^6 + \cdots - 1$$

$$>> p = \text{poly}([1 1 1 1 1 1 1])$$

$$p =$$

$$\begin{align*}
1 & -7 & 21 & -35 & 35 & -21 & 7 & -1 \\
\end{align*}$$

$$>> x = 0.99:.0001:1.01;$$

$$>> \text{plot}(x, \text{polyval}(p, x), \cdot \cdot \cdot)$$
Root-finding codes should have more than one stopping criterion (p. 81)

Stop the iteration if

- \(|x_{k+1} - x_k| \leq \tau_x\)
- \(|f(x_k)| \leq \tau_f\)
- \(k \geq k_{\text{max}}\)

\(k_{\text{max}}\) prevents infinite loops such as this one (bug in Matlab 7.1)

\[
>> \text{fzero(@sin,[3 4],optimset('tolx',1e-20))}
\]
what happened, what’s next

• iteration converges if $|\varphi'| \leq m < 1$

• Newton-Raphson has quadratic convergence to simple roots

• error estimation formula

• achievable accuracy

• three stopping criteria

Next lecture: interpolation (§5.1-4)