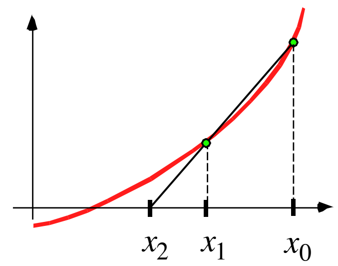
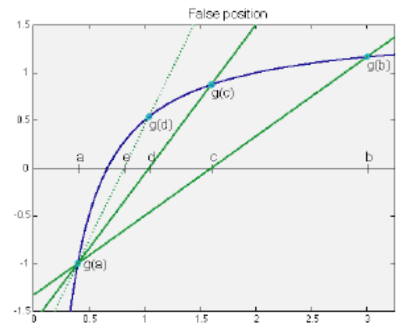
Finding roots

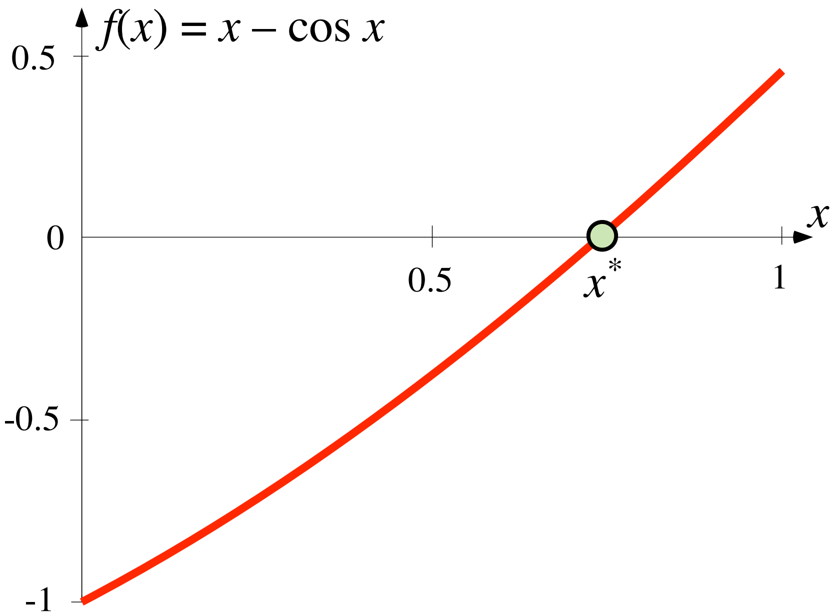
- introducing the problem
- bisection method
- Newton-Raphson method
- secant method
- fixed-point iteration method





We need methods for solving nonlinear equations

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



Numerical methods are used when

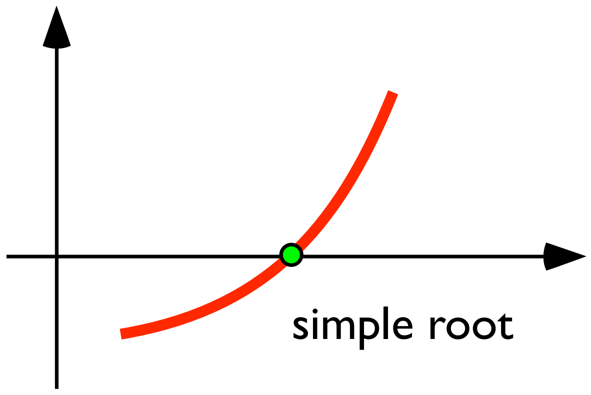
- there is no formula for root,
- the formula is too complex,
- f is a "black box"

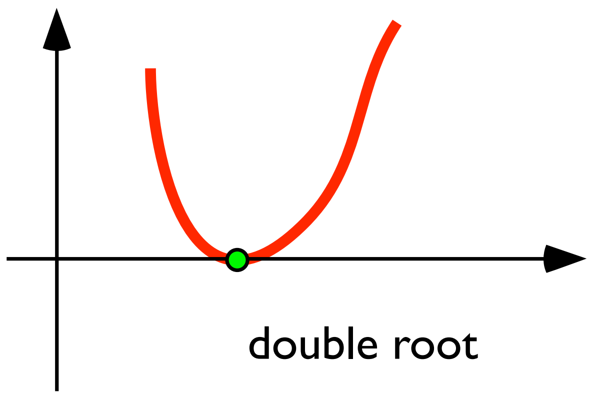
Roots can be simple or multiple

*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

use theory

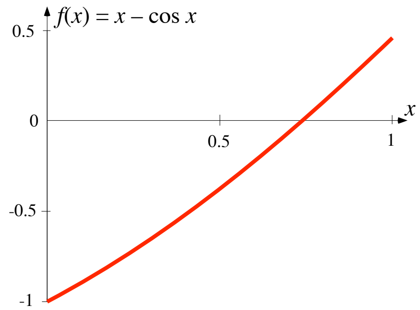
All roots of $x - \cos x = 0$ lie in the interval $[-1,1]$

Proof:

$x = \cos x$

$\Rightarrow \quad |x| = |\cos x| \leq 1$

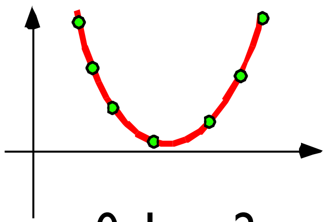
use graphics



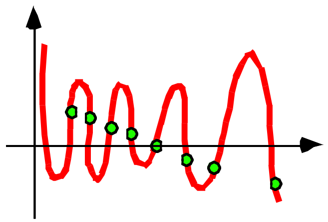
3 rules

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

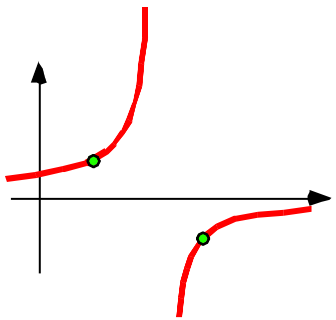
difficult cases:



0, 1 or 2 roots?



many roots



pole

Bisection traps a root in a shrinking interval

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 $x=\dfrac{a+b}{2}$ & $\text{sign}(f(x))$;

repeat using $[a,x]$ or $[x,b]$ as new bracketing interval.

```
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
```

```
>> f = @(x) x-cos(x);
>> bisection(f,0.7,0.8,1e-3);

   a             b          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
```

Bisection is slow but dependable

Advantages

- guaranteed convergence
- predictable convergence rate
- rigorous error bound

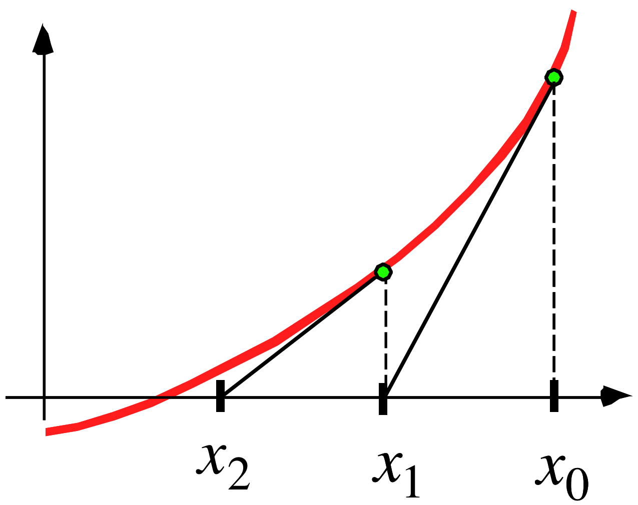
Disadvantages

- may converge to a pole
- needs bracketing interval
- slow

Newton-Raphson method uses the tangent

Iteration formula

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




```
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)\f(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);

k     x_k                 f(x_k)     f'(x_k)       dx
0    0.700000000000    -6.48e-02    1.64422    -0.039436497848
1    0.739436497848     5.88e-04    1.67387     0.000351337383
2    0.739085160465     4.56e-08    1.67361     0.000000027250
3    0.739085133215     2.22e-16    1.67361     0.000000000000
```

Newton-Raphson is fast

Advantages

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

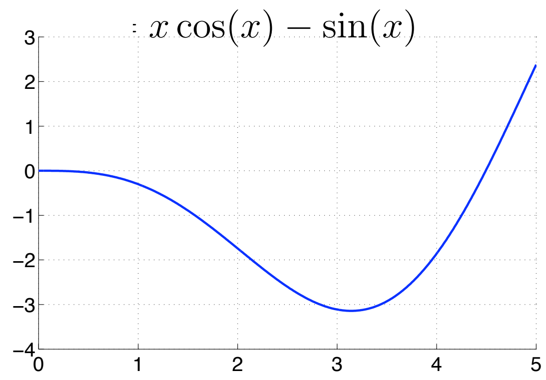
Disadvantages

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

Exercise

Find the first positive root of

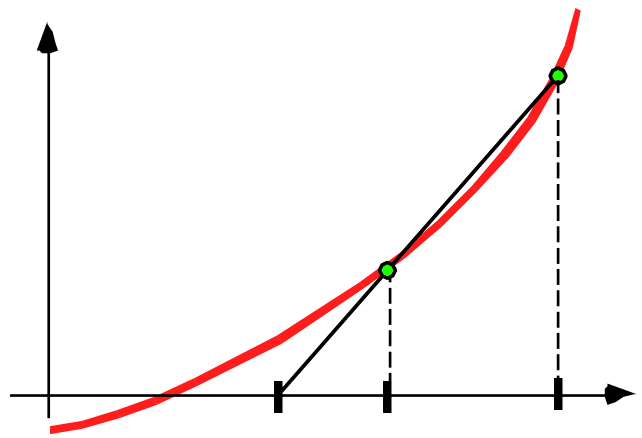
$x = \tan x$



Secant method is derivative-free

Iteration formula

$$x_{k+1} = x_k - \frac{f(x_k)}{\left(\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}\right)}$$




```
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);

k    x_k                   f(x_k)
0   0.70000000000000   -6.48422e-02
1   0.80000000000000    1.03293e-01
2   0.73856544025090   -8.69665e-04
3   0.73907836214467   -1.13321e-05
4   0.73908513399236    1.30073e-09
5   0.73908513321516   -1.77636e-15
6   0.73908513321516    0.00000e+00
```

Secant method is also fast

Advantages

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

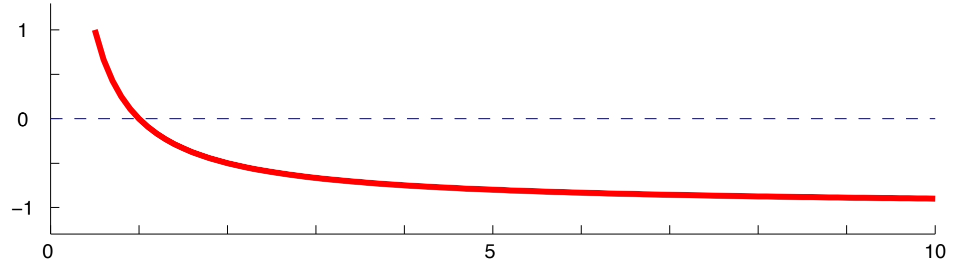
Disadvantages

- iterates may diverge
- no practical & rigorous error bound

Without bracketing, an iteration can jump far away

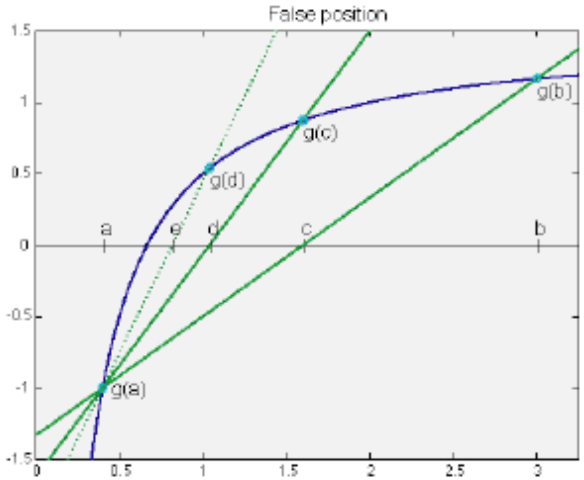
Example

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

k          x_k              f(x_k)
0          0.5     1.00000e+00
1           10    -9.00000e-01
2          5.5    -8.18182e-01
3        -39.5    -1.02532e+00
4       183.25    -9.94543e-01
```



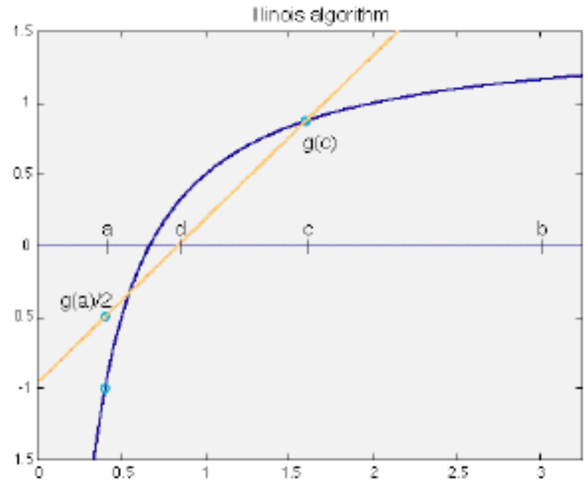

```
>> df = @(x) -1/x^2;
>> newtonraphson(f,df,10,3);

k             x_k        f(x_k)      f'(x_k)              dx
0              10     -9.00e-01    -0.01000              90
1             -80     -1.01e+00    -0.00016            6480
2           -6560     -1.00e+00    -0.00000     4.30402e+07
3    -4.30467e+07     -1.00e+00    -0.00000     1.85302e+15
```

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
```



Illinois method halves function value whenever endpoint is re-used: it is fast and reliable

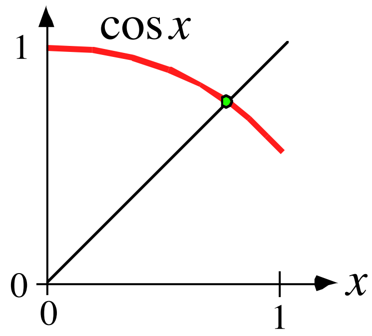
Brent's method combines bisection, secant and inverse quadratic interpolation

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

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

Zero found in the interval [0.5, 10]

x =

   1.00000000002072
```

Root-finding can be treated as fixed-point-finding

Fixed point problem

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



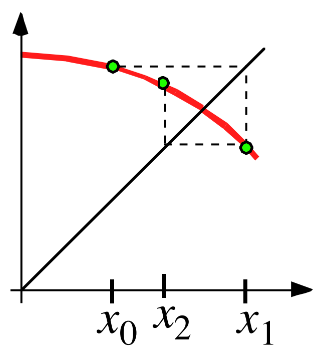
Fixed-point iteration

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

Example (p. 71)

enter 0.7 on your calculator

press ***cos*** repeatedly



Example (p.72)

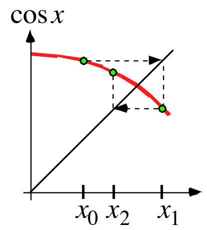
instead, press ***arccos*** repeatedly

Newton-Raphson method is also a fixed point iteration (p. 70)

$$x_{k+1} = \underbrace{x_k - \frac{f(x_k)}{f'(x_k)}}_{\varphi(x_k)}$$

what happened

- 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



- convergence
- error estimate & achievable accuracy
- stopping criteria