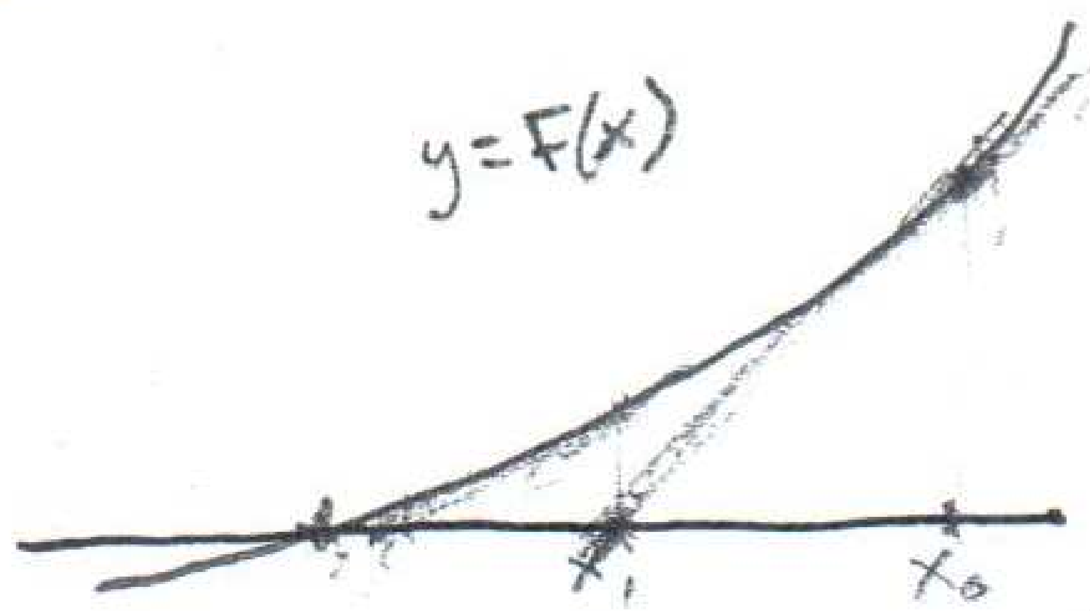
Root finding: Newton-Raphson Method for $F(x) = 0$

Idea: Start with an initial guess x_0



construct tangent line at $(x_0, F(x_0))$,
find x-intercept of line

$$y - F(x_0) = F'(x_0)\cdot(x - x_0)$$

$$\overset{y}{\underset{0}{\shortparallel}} \Rightarrow x = x_0 - \frac{F(x_0)}{F'(x_0)}$$

repeat

From Taylor expansion about x_0:

$$F(x_0 + \Delta x) \approx F(x_0) + F'(x_0)\cdot\Delta x$$

$$\underset{0}{\text{// want}} \quad \Rightarrow \Delta x = -\frac{F(x_0)}{F'(x_0)}$$

$$x = x_0 + \Delta x$$

repeat

Need $F(x), F'(x)$ continuous near root

Newton-Raphson algorithm for root of $F(x)=0$: initial guess x_0 (near root r)

Newton step: $\Delta x = -\dfrac{F(x_n)}{F'(x_n)}$

$x_{n+1} = x_n + \Delta x$, $n = 0, 1, \ldots, \text{maxIT}$

Stop when $|\Delta x| < TOL$ **and** $|F(x_n)| < TOL$

Convergence of Newton: If $F \in C^1$ near a simple root r ($F(r) = 0$ but $F'(r) \neq 0$) and if initial guess x_0 is "sufficiently close" to r then Newton iterates $\{x_n\}$ converge to r quadratically.

Quadratic order of convergence: $|x_{n+1} - r| \leq C|x_n - r|^2$

$$|e_{n+1}| \leq C\cdot|e_n|^2$$

amounts to (roughly) doubling the number of correct binary digits at each iteration!

Very fast convergence!

Newton-Raphson root finder

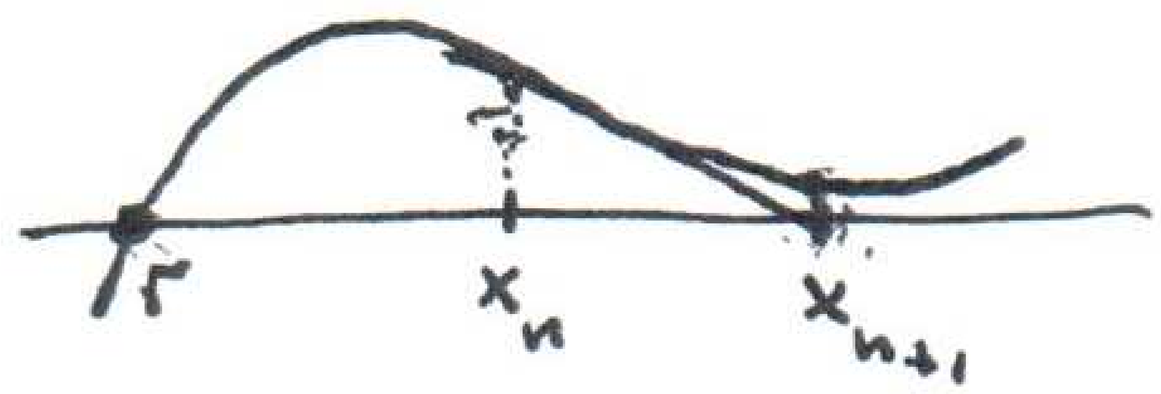
Advantages: 1. converges <u>quadratically</u> fast <u>if</u> it converges, very fast!

2. applicable in any dimension.

Disadvantages: 1. <u>Not robust</u>: may fail to converge, unless x_0 very close to root.

2. Requires formula for $F'(x)$, and be able to evaluate at any x_n, which may be unavailable and/or expensive.

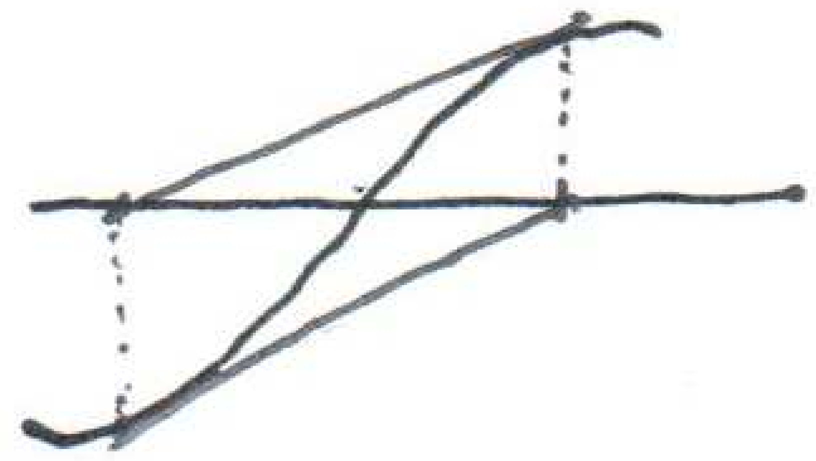
3. May give false root: $|F(x_n)| \approx 0$



must check both $|F(x_n)| < TOL$

<u>and</u> $|\Delta x| < TOL$

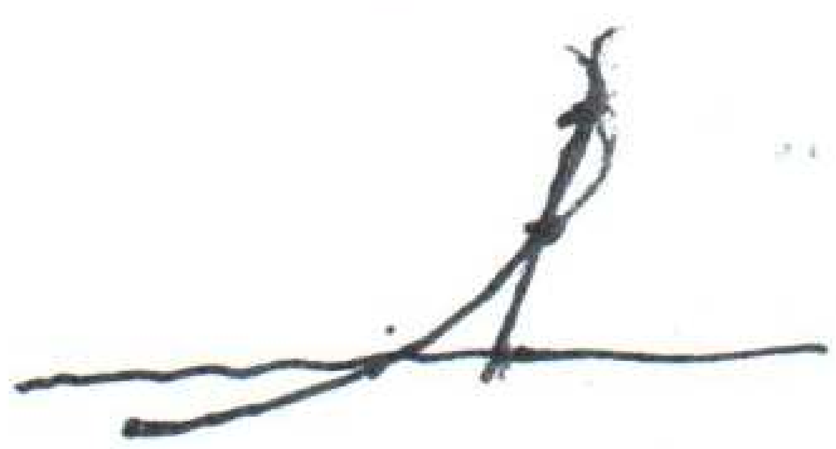
4. May converge to another root or get stuck:



<u>Remedy for 2.</u> Use "Secant Method": replace $F'(x_n) \approx \dfrac{F(x_n)-F(x_{n-1})}{x_n - x_{n-1}}$

(needs two starting pts)

Geometrically: replace tangent by secant:



Order of convergence is $\frac{1+\sqrt{5}}{2} \approx 1.62$, less than quadratic but still superlinear.

Newton-Raphson is a fixed point iteration: $x = g(x) := x - \dfrac{F(x)}{F'(x)}$

Finding $\sqrt{a}$: Solve $F(x) = x^2 - a = 0$, $a > 0$.

via Newton: $x_{n+1} = x_n - \dfrac{x_n^2 - a}{2x_n} = \frac{1}{2}\left(x_n + \dfrac{a}{x_n}\right)$ an ancient method!

iteration of $x = g(x) = \frac{1}{2}\left(x + \frac{a}{x}\right)$

e.g. $\sqrt{4}$: solve $x^2 - 4 = 0$

$x_0 = 1$, $x_1 = \frac{1}{2}\left(1 + \frac{4}{1}\right) = \frac{5}{2}$, $x_2 = \frac{1}{2}\left(\frac{5}{2} + 4 \cdot \frac{2}{5}\right) = \frac{41}{20} \approx 2.05$

$x_3 = \frac{1}{2}\left(\frac{41}{20} + \frac{4 \cdot 20}{41}\right) = \frac{1}{2} \cdot \frac{41^2 + 4 \cdot 20^2}{20 \cdot 41} \approx 2.000609756$

e.g. Golden Ratio φ is root of $F(x) = x^2 - x - 1 = 0$:

$$x_{n+1} = x_n - \underline{\qquad\qquad} = \frac{x_n^2 + 1}{2x_n - 1} = \frac{x_n + \frac{1}{x_n}}{2 - \frac{1}{x_n}}$$

$x_0 = 1$, $x_1 = 2$, $x_2 = \frac{5}{3} \approx 1.66$, $x_3 = \frac{34}{21} \approx 1.619$, $x_4 = 1.618$, ...

Other root finding methods: Müller
Inverse Quadratic Interpolation
...

Root finder codes: ZEROIN(a, b, TOL) excellent! uses Brent's method:
bisection in [a, b], then Inverse Interpolation

Matlab has fzero: [root, Feval] = fzero(F_handle, Init)

F_handle = @FCN with y=FCN(x) coded in FCN.m

Init = x0 initial guess for target root

= [a, b] interval containing target root (better)

Newton algorithm → code

$$\Delta x = -\frac{F(x_n)}{F'(x_n)}$$

$$X_{n+1} = X_n + \Delta x$$

```
inputs :  x0 , maxIT , TOL

initialize :  xn = x0 ,  Dx = 10 (something big)

print labels for columns of iterates :  n   xn   Fn

for n = 1 : maxIT
    evaluate F, F' :  [Fn , DFn] = FCN(xn)
    print iterate :  n , xn , Fn

    Test for convergence: both Dx and residual:
        if |Dx| < TOL · |xn|
            if |Fn| < TOL
              DONE :  root = xn found in n iters , residual = Fn
              break
            else
              Stuck! at iter n , Dx < TOL but Fn > TOL , exiting
              break
            end
        end
    Perform Newton step :
        Dx = - Fn / DFn
        xn = xn + Dx
end for-Loop

if n ≥ maxIT
    print :  Out of iters , [redacted] try bigger maxIT ...
end
```

y = F(x) coded in FCN.m :

```
function[y, Dy] = FCN(x)
    y = x^2 - x - 1;
    Dy = 2*x - 1;
end
```