ACM40290: Numerical Algorithms

Algorithms for Solving Nonlinear Equations

Dr Barry Wardell
School of Mathematics and Statistics
University College Dublin

Algorithms for Solving Nonlinear Equations

- Iterative algorithms for non-linear equations fall into two broad categories :
    1. Locally convergent and fast.
    2. Globally convergent and slow.
- A good algorithm should:
    1. Be easy to use, preferably using only information on f, not on its derivative.
    2. Be reliable, i.e., it should find a root close to an initial guess and not go off to become chaotic.
- There is no ideal method. MATLAB uses a combination of methods to find the root. We will study a few of these.

Algorithms for Solving Nonlinear Equations

- Bisection algorithm
- Newton's (Newton-Raphson) method
- Secant algorithm
- Multipoint secant algorithms (e.g. Muller's three-point algorithm, inverse quadratic interpolation)

The Bisection Algorithm

- The idea behind this method is to

  1. Find an interval $[x_1, x_2]$ over which f changes sign, i.e., $\text{sign}[f(x_1)] \neq \text{sign}[f(x_2)]$, and calculate $x_3 = (x_1 + x_2)/2$.

  2. Then f must change sign over one of the two intervals $[x_1, x_3]$ or $[x_3, x_2]$.

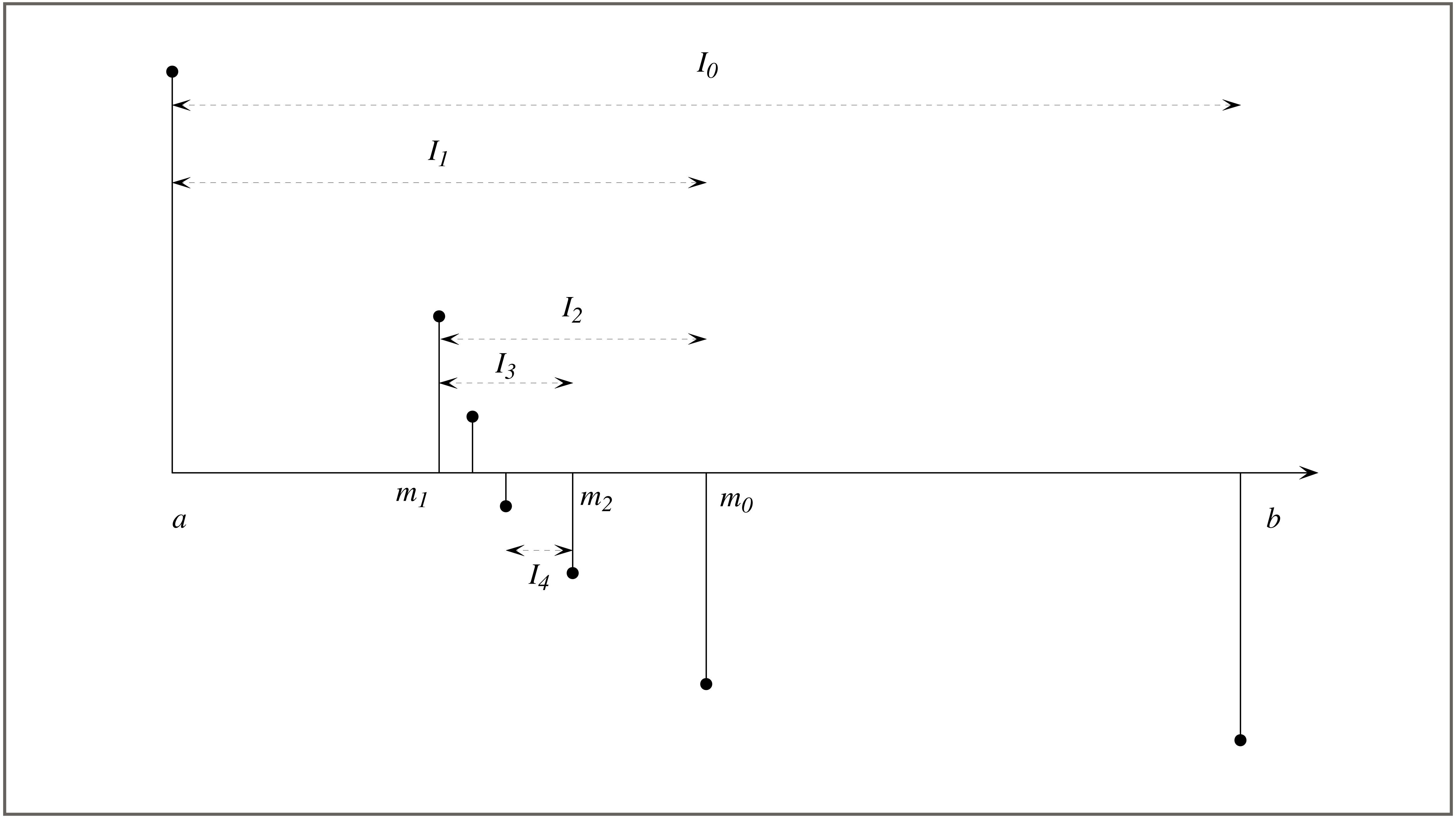
  3. Replace $[x_1, x_2]$ by the bisected interval over which f changes sign.

  4. Repeat until the interval is smaller than some specified tolerance.

- Notice that the value or shape of the function plays no part in the algorithm. Only the sign is of interest.

- This is a reliable method as the interval over which the solution is known reduces in size by a factor of at least two at each iteration. However it is very slow, and its convergence is linear.

The Bisection Algorithm



Implementation of the Bisection Algorithm

```
algorithm  Bisect(f, a, b, ϵ, maxits)
-----------------------------------------
    f_a := f(a); f_b := f(b)
    for  k := 1  to  maxits  do
        m := (a + b)/2
        f_m := f(m)
        if  f_m = 0  then  return  (m)
        else if  sign(f_m) = sign(f_a)  then
            a := m
            f_a := f_m
        else
            b := m
            f_b := f_m
        endif
        if  |a − b| ≤ ϵ  then  return  (a + b)/2
    endfor
endalg  Bisect
```

Analysis of the Bisection Algorithm

- The error decreases by a factor of 2 at each iteration

$$e_k = \frac{1}{2} e_{k-1}$$

- The error after k iterations is $e_k = 2^{-k}\,|\,b\text{-}a\,|$. The algorithm stops after k iterations with $2^{-k}\,|\,b\text{-}a\,| = \epsilon$, or $2^{k}\,\epsilon = |\,b\text{-}a\,|$, or $2^{k} = e_0/\epsilon$. So,

$$k = \left\lceil \log_2 \frac{e_0}{\epsilon} \right\rceil$$

- If the algorithm starts with $e_0 = 2^{-1}$ then after 52 iterations the error is $e_{52} = 2^{-53} = 10^{-16}$, i.e. IEEE double precision.

- The main work of the algorithm is the evaluation of $f\,(\cdot)$. Although it may appear that 2 function evaluations are needed per iteration, only one, $f(m)$, is needed if $f(a)$ or $f(b)$ is saved between iterations. Thus bisection performs about 53 function evaluations to obtain full precision.

Bisection Example: $\sqrt{c}$

$$x = \sqrt{c}$$

$$f(x) = x^2 - c = 0$$

Example: $c = 4$, $a_0 = 1.8$, $b_0 = 2.8$

k	0	1	2	…	6	…	52
a_k	1.8	1.8	1.8	…	1.9875,		2.000000
b_k	2.8	2.3	2.05		2.00313		2.000000
e_k	1	0.5	0.25	…	0.015625		$2.22*10^{-16}$

Newton's Method

The idea behind this method is:

- Evaluate f and f' at x_1;
- Approximate f by a line of slope f' through the point $(x_1, f(x_1))$;
- Find the point x_2 where this line crosses zero so that

$$x_2 = x_1 - f(x_1)/f'(x_1)$$

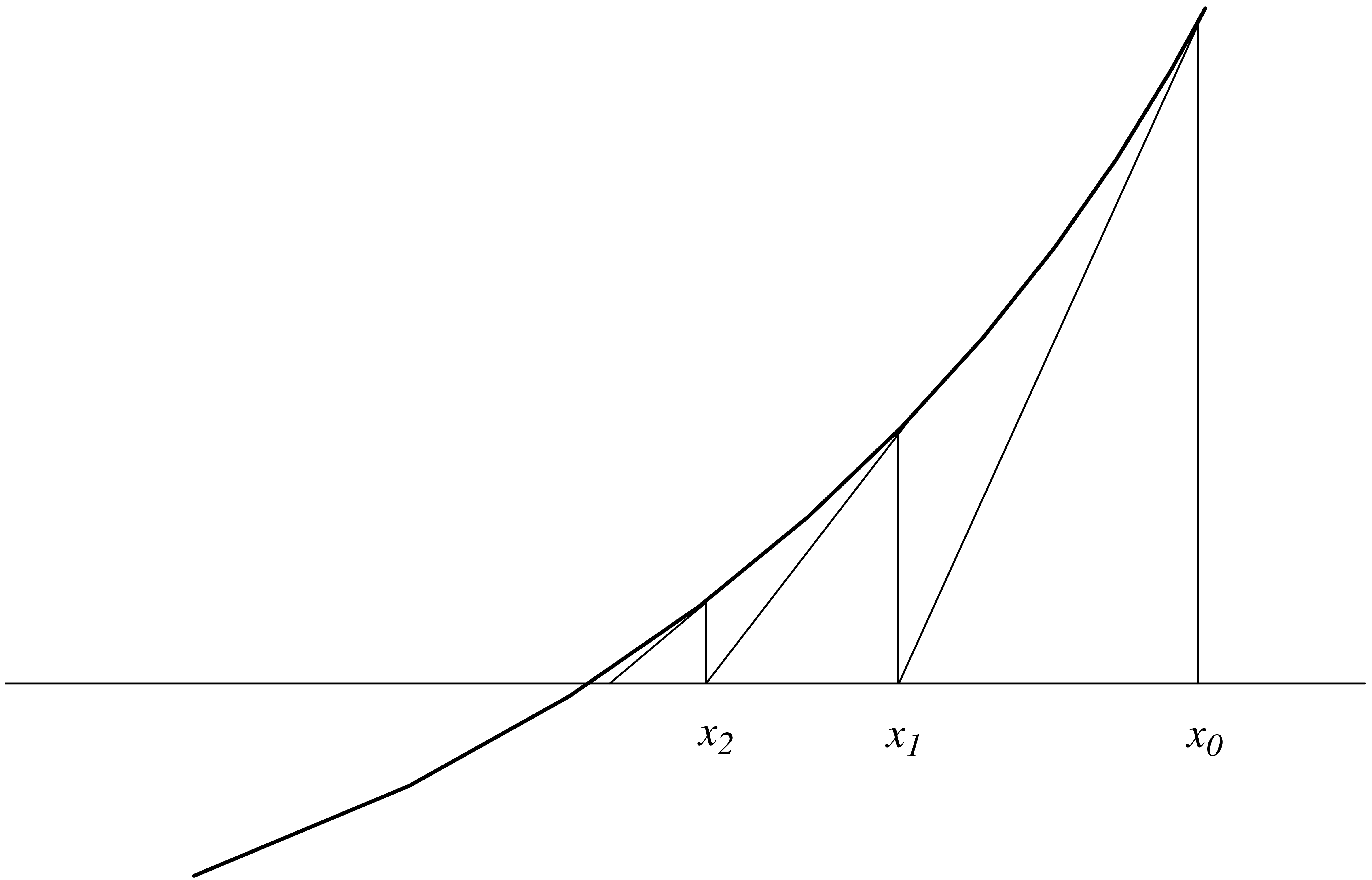
- Replace x_1 by x_2 and continue to generate a series of iterations x_i. Stop when

$$|f(x_1)| < \text{TOL} \quad \text{or when} \quad |x_{i+1} - x_i| < \text{TOL}$$

- For a general iteration k, Newton's method is

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

Newton's Method



Implementation of Newton's Method

algorithm $\texttt{Newton}\ (f, f', x_{\texttt{old}}, \epsilon, \texttt{maxits})$

$f_{\texttt{old}} = f(x_{\texttt{old}})$; $f'_{\texttt{old}} := f'(x_{\texttt{old}})$

for $k := 1$ **to** $\texttt{maxits}$ **do**

    $x_{\texttt{new}} := x_{\texttt{old}} - f_{\texttt{old}} / f'_{\texttt{old}}$

    $f_{\texttt{new}} := f(x_{\texttt{new}})$; $f'_{\texttt{new}} := f'(x_{\texttt{new}})$

    if $|x_{\texttt{new}} - x_{\texttt{old}}| \leq \epsilon$ OR $f_{\texttt{new}} = 0$ **then**

        return $(x_{\texttt{new}})$

    else

        $x_{\texttt{old}} := x_{\texttt{new}}$

        $f_{\texttt{old}} := f_{\texttt{new}}$; $f'_{\texttt{old}} := f'_{\texttt{new}}$

    endif

endfor

endalg Newton

Analysis of Newton's Method

- Newton's method is very fast and generalises to higher dimensions in a straightforward way **but** it needs derivatives which may be hard to compute.
- Indeed, we may simply **not** have derivatives of f, for example, if f is an experimental measurement.
- Newton's method often requires x_1 to be close to the root x^* to behave reliably.
- When Newton's method works, $|e_{n+1}| \approx K|e_n|^2$ and the convergence is quadratic in the number of iterations.

Convergence of Newton's Method

$$f(x) = 0 \;\Rightarrow\; x = T(x)$$

We get the order of convergence by examining the derivatives of T(x) at the fixed point.

$$T(x) = x - f(x)/f'(x)$$

At a fixed point,

$$\begin{aligned} x = T(x) \;&=\; x - f(x)/f'(x) \\ &\Rightarrow f(x)/f'(x) = 0 \\ &\Rightarrow f(x) = 0 \text{ if } f'(x) \neq 0 \end{aligned}$$

Convergence of Newton's Method

The first derivative is

$$T'(x) = 1 + \frac{f''(x)f(x)}{[f'(x)]^2} - \frac{f'(x)}{f'(x)} = f(x)\left(\frac{f''(x)}{[f'(x)]^2}\right)$$

At a fixed point $f(x) = 0$ and so $T'(x) = 0$. Hence Newton's method has at least 2nd order convergence. The second derivative of $T(x)$ is

$$T''(x) = f(x)\left(\frac{f'''(x)}{[f'(x)]^2} - \frac{2[f''(x)]^2}{[f'(x)]^3}\right) + \frac{f''(x)}{f'(x)} = \frac{f'''(x)}{f'(x)} \neq 0, \quad \text{at } f(x) = 0.$$

So, in general Newton's algorithms is 2nd order, i.e.,

$$e_k = ce_{k-1}^2 \quad \Rightarrow \quad e_k = ce_0^{2^k}$$

Convergence of Newton's Method

The algorithm stops after k iterations with $e_k = e_0^{2^k} = \epsilon$. Taking logs of both sides we get

$$2^k \log_2 e_0 = \log_2 \epsilon \quad \text{or} \qquad 2^k = \frac{\log_2 \epsilon}{\log_2 e_0}$$

Again, taking logs of both sides we get

$$k = \left\lceil \log_2 \frac{\log_2 \epsilon}{\log_2 e_0} \right\rceil$$

Hence, with $e_0 = 2^{-1}$ and $\epsilon = 2^{-53} \approx 10^{-16}$ we need

$$n = \left\lceil \log_2 \frac{\log_2 2^{-}53}{\log_2 2^{-}1} \right\rceil = \lceil \log_2 53 \rceil = 6 \quad \text{iterations}$$

Thus we get full IEEE double precision after 6 iterations. **This rapid convergence comes at a price.**

Newton's Method Example: $\sqrt{c}$

$$x = \sqrt{c}$$

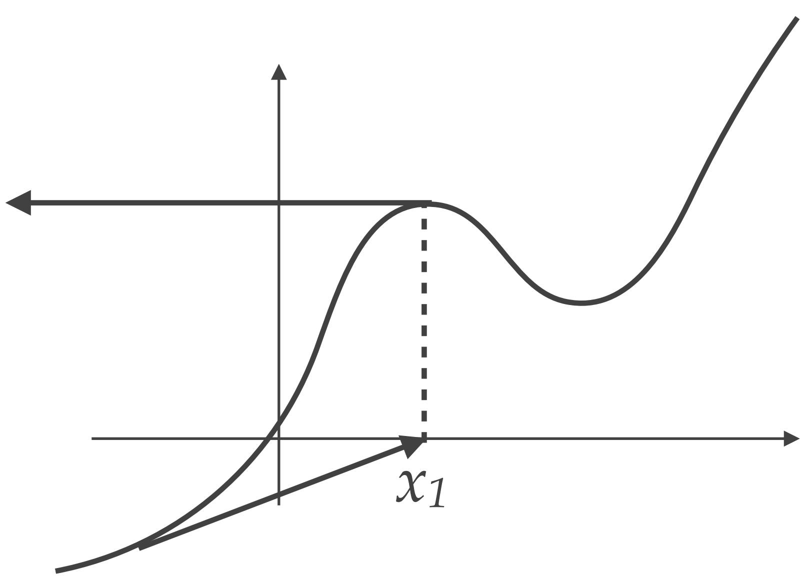
$$f(x) = x^2 - c = 0$$

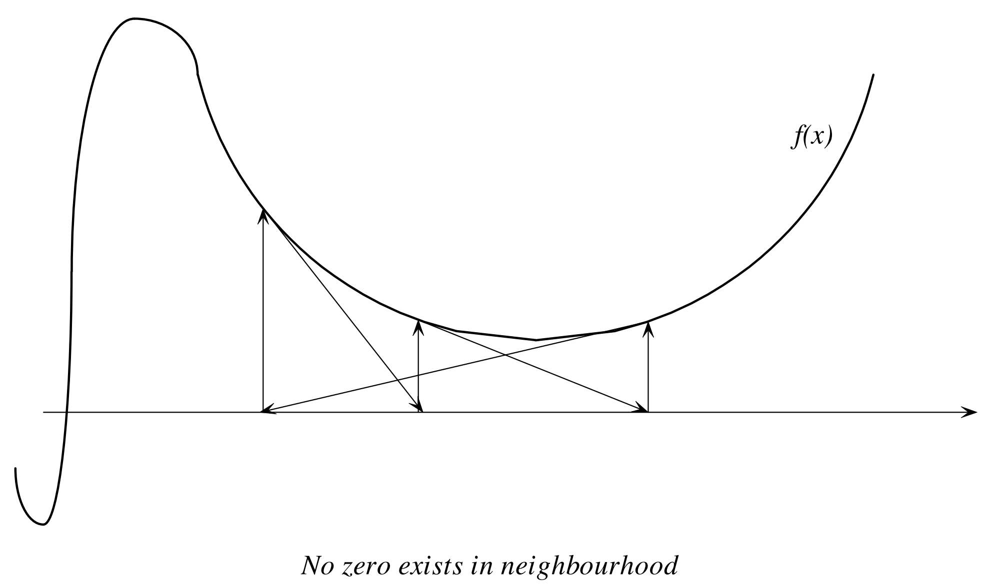
Example: $c = 4$, $x_0 = 1.0$

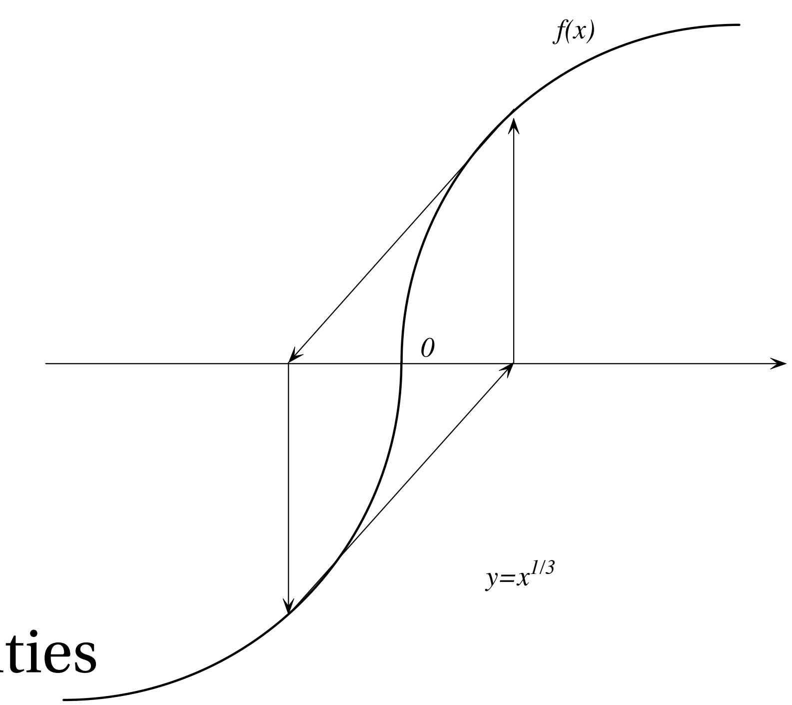
k	0	1	2	3	4	5	6
x_k	1.0	2.5	2.05	2.0006098	2.0000001	2.0000000	2.0000000
e_k	1	0.5	0.05	0.000609	9.2×10^{-8}	2.15×10^{-15}	1.16×10^{-30}

Difficulties with Newton's Method

1. The function $f(x)$ and its derivative must be evaluated at each iteration. Also, we need to find and program the derivative. Finding the derivative may be difficult, if not impossible. For example, the value of $f(x_k)$ may be the result of a long numerical simulation or an experiment.
2. The convergence of Newton's algorithm is local and so good starting values are needed if convergence is to occur at all.







The Newton Algorithm in Difficulties

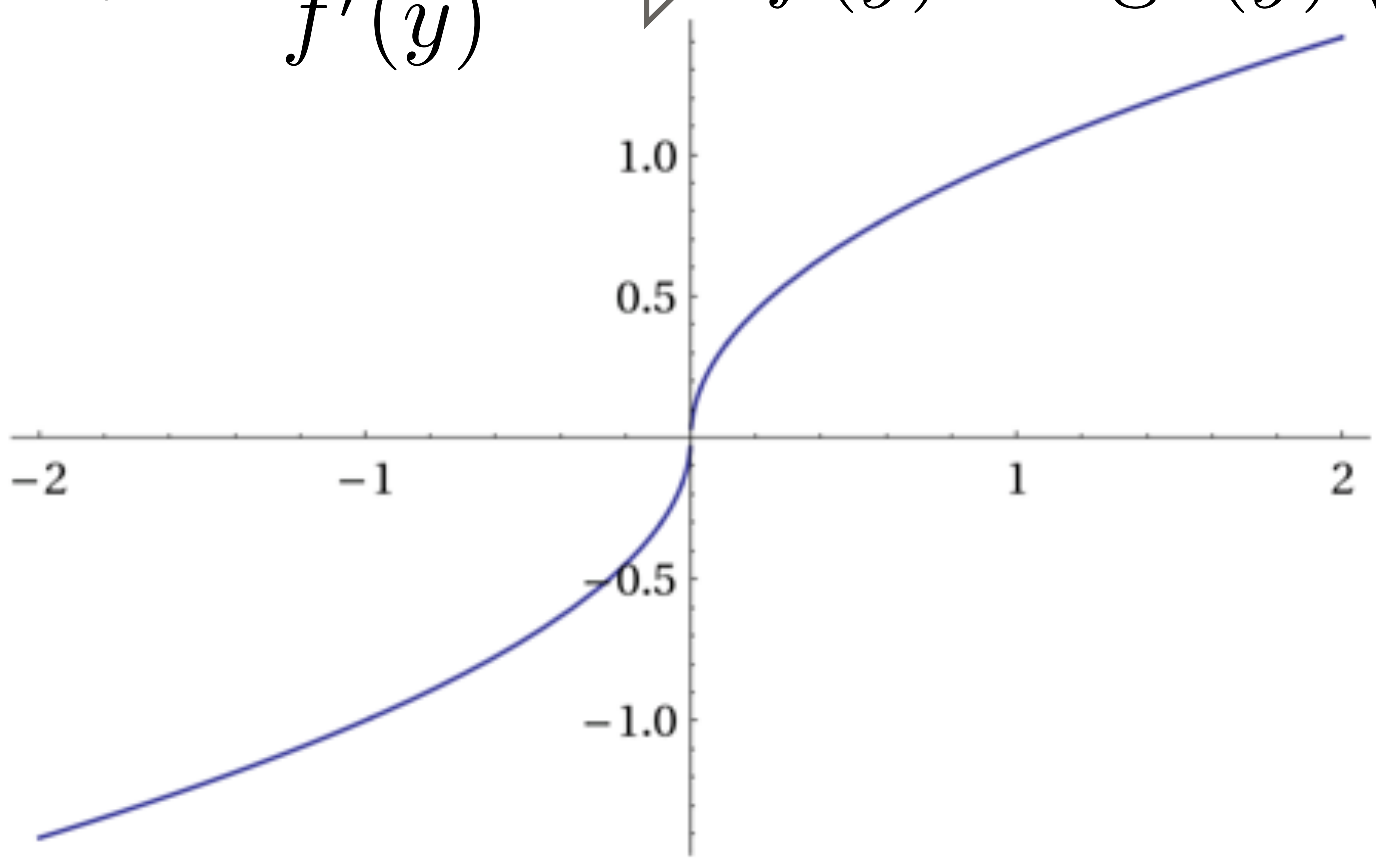
Difficulties with Newton's Method

Cycles

Consider f that gives $x_n = y$, $x_{n+1} = -y$. Newton's method is

$$-y = y - \frac{f(y)}{f'(y)}$$

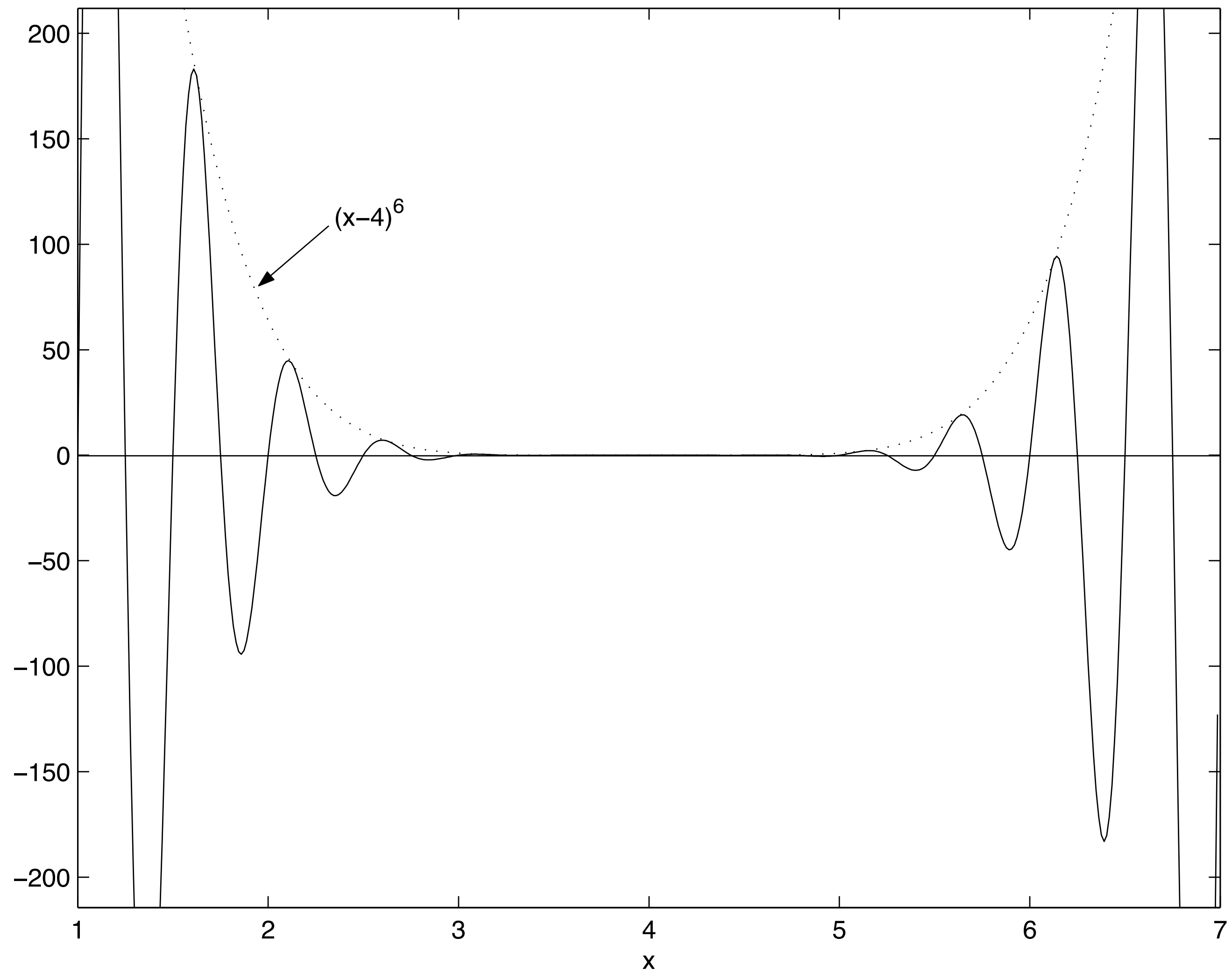
$$2y = \frac{f(y)}{f'(y)} \Rightarrow f(y) = \operatorname{sgn}(y)\sqrt{|y|}$$



Difficulties with Newton's Method

Convergence to a Multiple Zero

If $f(x)$ has the form $f(x) = (x - a)^p\, g(x)$, then this function has a *zero of multiplicity p at a*.



$$f(x) = (x-4)^6 \sin 4\pi x$$

Newton's Method

Convergence to a Multiple Zero

Using $f(x) \approx (x\text{-}a)^p\, g(x)$, we get the iteration mapping

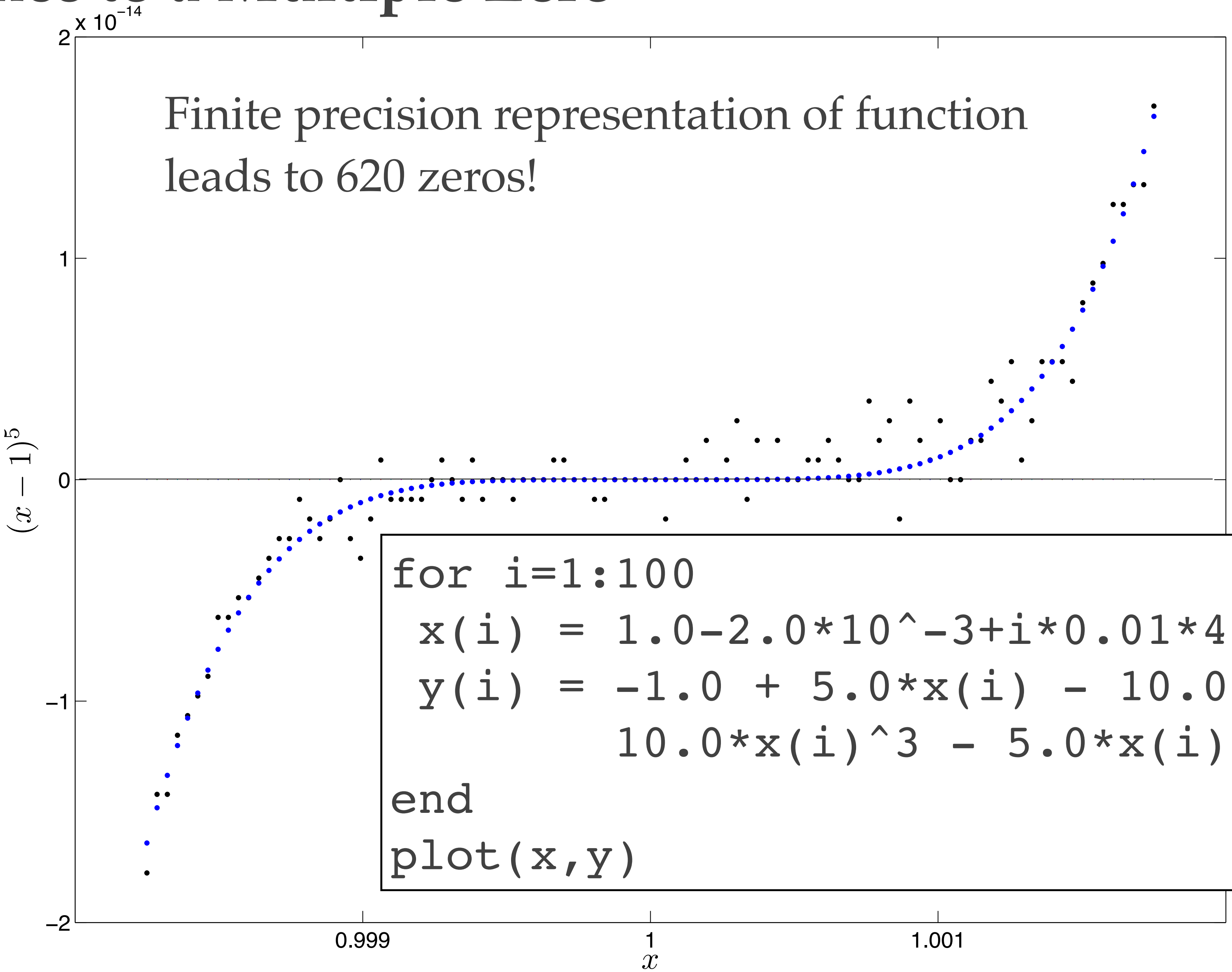
$$T(x) = x - \frac{f(x)}{f'(x)}$$
$$= x - \frac{(x-a)^p}{p(x-a)^{p-1}} = x - \frac{x-a}{p}$$
$$T'(x) = 1 - \frac{1}{p}$$

This shows that $T'(x) \neq 0$ for $p \neq 1$ and so Newton's algorithm has 1st order convergence at a multiple zero.

$$e_{k+1} \approx T'(x_k)e_k = \left(1 - \frac{1}{p}\right)e_k$$

Difficulties with Newton's Method

Convergence to a Multiple Zero




```
for i=1:100
 x(i) = 1.0-2.0*10^-3+i*0.01*4*10^-3;
 y(i) = -1.0 + 5.0*x(i) - 10.0*x(i)^2 +...
          10.0*x(i)^3 - 5.0*x(i)^4 + x(i)^5;
end
plot(x,y)
```

$$f(x) = (x-1)^5 = x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1$$

The Secant Method

The Secant method is the same as Newton's method, but replacing the exact derivative with a finite difference approximation:

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

Then, the **Secant** iteration formula is

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

Implementation of the Secant Method

```
algorithm Secant (f, x0, x1, ε, maxits)
------------------------------------------------
    f0 := f(x0);  f1 := f(x1)
    for k := 1 to maxits do
        x2 := x1 − f1 ⋆ (x1 − x0)/(f1 − f0)
        f2 := f(x2)
        if |x2 − x1| ≤ ε OR f2 = 0 then return (x2)
        else
            x0 := x1;  f0 := f1
            x1 := x2;  f1 := f2
        endif
    endfor
endalg Secant
```

Analysis of the Secant Method

Derivation of the order of convergence for the Secant method is long and tedious (see Conte & De Boor, pg. 103 for details). We're only interested in the result, which is

$$e_{k+1} \approx c e_k^{1.618\ldots}$$

$\dfrac{1+\sqrt{5}}{2}$
(Golden ratio)

Question

How does the Secant method behave for $f(x) = \mathrm{sgn}(x)\sqrt{|x|}$?

Analysis of the Secant Method

Error after k iterations

$$e_k = e_0^{p^k} \qquad (e_k = e_{k-1}^p)$$

Stop when

$$e_0^{p^k} = \epsilon \implies p^k \log_2 e_0 = \log_2 \epsilon \implies p^k = \frac{\log_2 \epsilon}{\log_2 e_0}$$

$$\implies k = \left\lceil \frac{\log_2 \left(\frac{\log_2 \epsilon}{\log_2 e_0} \right)}{\log_2 p} \right\rceil$$

Analysis of the Secant Method

If $e_0 = 2^{-1}$, $\epsilon = 2^{-53} \approx 10^{-16}$

$$k = \left\lceil \frac{\log_2 53}{\log_2 1.618} \right\rceil = 9$$

Slower convergence than Newton's method, but requires just one function evaluation per iteration.

[Full IEEE precision: Newton 12 function calls
Secant 9 function calls

Root finding in MATLAB

Matlab combines these methods into a single instruction,

```
x = fzero('fun', xguess)
```

or

```
x = fzero('fun', xguess, optimset('TolX', 1e-10))
```

It uses a combination of: Bisection, Secant, and Inverse Quadratic Interpolation. Searches for an interval in which the function changes sign. Robust, including checks for infinites, NaNs, complex numbers, etc.

History: Dekker, Math. Centre Amsterdam (1960s)
Brent (1973)

Root finding in MATLAB

Example: Find zero of

$$f(x) = \sin x - \frac{x}{2}$$

fun1.m

```
function output = fun1(x)
  output = sin(x) - x/2;
```

```
xguess = input('initial guess');
tol = input('error tolerance for the solution');
options = optimset('Display', 'iter', 'TolX', tol);

fzero('fun1', xguess, options)
```

Display each iteration

Multipoint Secant Methods

Müller's Quadratic Interpolation Algorithm

This algorithm uses the quadratic $a_0 + a_1 x + a_2 x^2$ to interpolate $f(x)$ at three points x_0, x_1, and x_2. The coefficients are found by solving

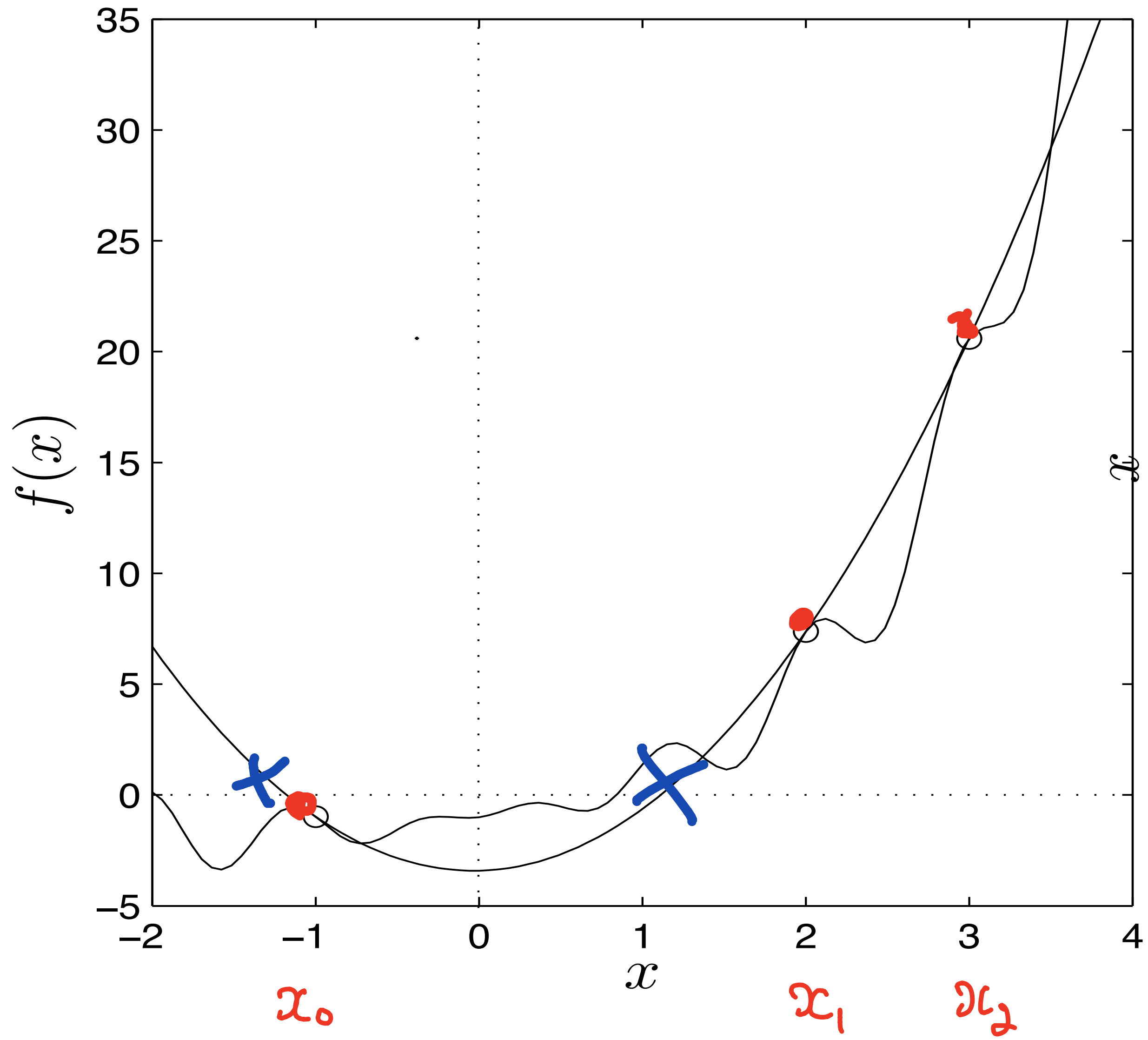
$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

The quadratic equation $a_0 + a_1 x + a_2 x^2$ is then solved to get a new point

$$x_{k+1} = \frac{-a \pm \sqrt{a_1^2 - 4a_0 a_2}}{2a_2} = T(x_k, x_{k-1}, x_{k-2})$$

Problem: there is a choice of two points, and the possibility that they are complex.

Multipoint Secant Methods



Two choices for x_3.

Then $x_4 = T(x_1, x_2, x_3)$

Multipoint Secant Methods

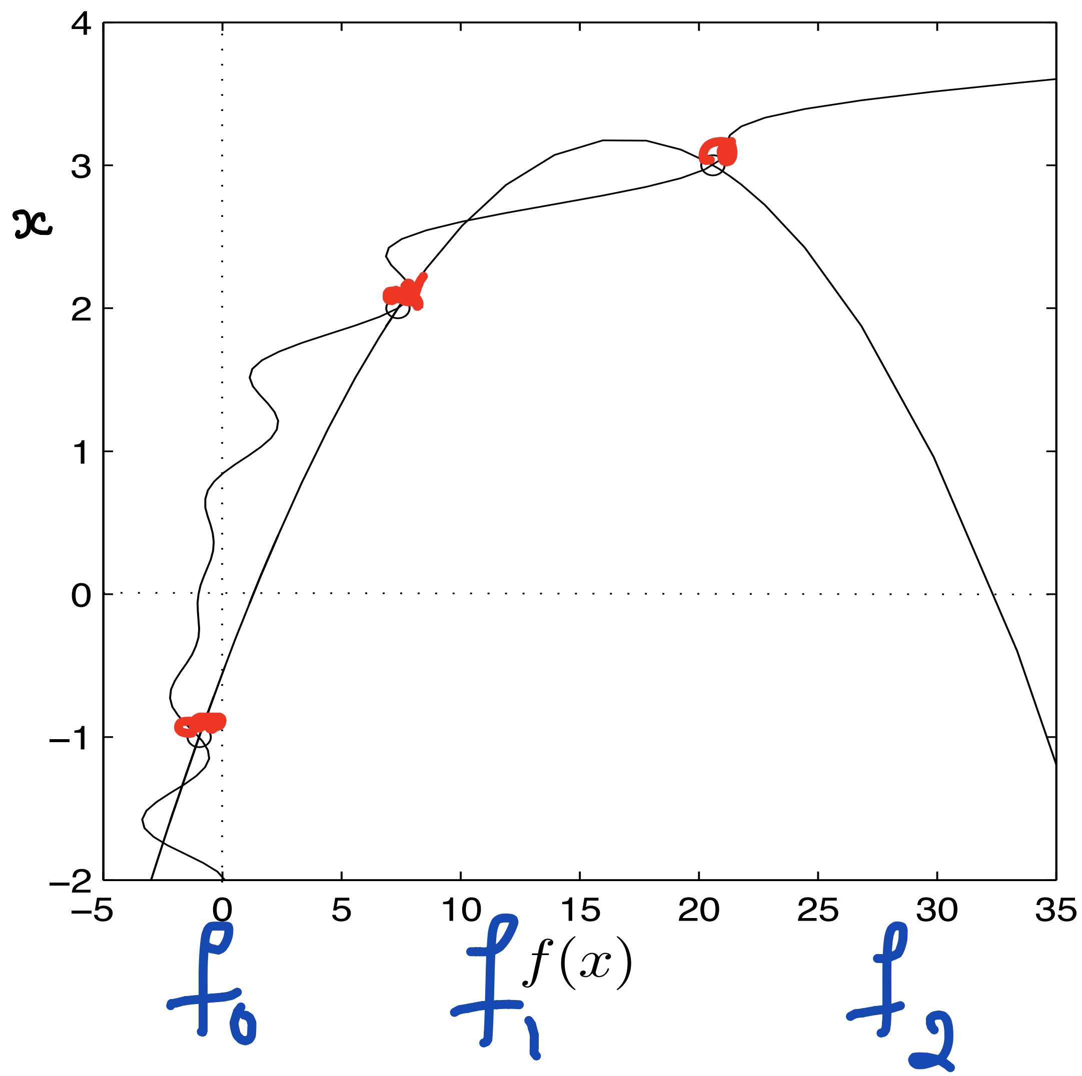
Inverse Quadratic Interpolation

Given $\{(x_0, f_0), (x_1, f_1), (x_2, f_2)\}$, we find the coefficients of the inverse quadratic function $p(f)$ that passes through the points $\{(f_0, x_0), (f_1, x_1), (f_2, x_2)\}$. This means we must solve

$$\begin{bmatrix} 1 & f_0 & f_0^2 \\ 1 & f_1 & f_1^2 \\ 1 & f_2 & f_2^2 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$$\Rightarrow x = p(f) = d_0 + d_1 f + d_2 f^2$$

Multipoint Secant Methods



We want the x that makes $f(x) = 0$.

The value of $p(f) = d_0 + d_1 f + d_2 f^2$ at $f = 0$ is d_0, i.e $x = d_0$.

Multipoint Secant Methods

Exercise: Derive an expression for d0 and use this to write a complete zero-finding algorithm.

Multipoint Secant Methods

Order of convergence

$$e_{k+1} \approx c e_k^{1.84}$$

Almost as good as Newton's method, but only need one function evaluation per iteration, provided previous ones are saved.