Numerical Solution of $f(x) = 0$

Gerald W. Recktenwald
Department of Mechanical Engineering
Portland State University
gerry@pdx.edu

Overview

Topics covered in these slides

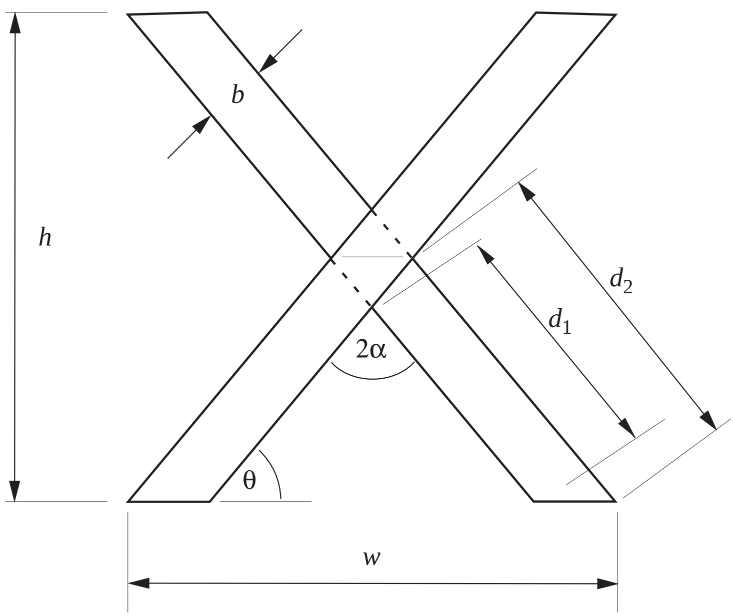
- Preliminary considerations and bracketing.
- Fixed Point Iteration
- Bisection
- Newton's Method
- The Secant Method
- Hybrid Methods: the built in `fzero` function
- Roots of Polynomials

Example: Picnic Table Leg

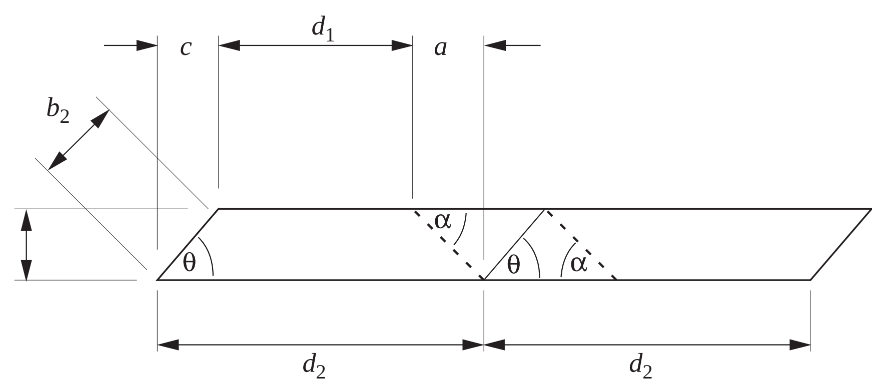
Example: Picnic Table Leg

Computing the dimensions of a picnic table leg involves a root-finding problem.

Leg assembly



Detail of one leg



Example: Picnic Table Leg

Dimensions of a the picnic table leg satisfy

$$w \sin\theta = h \cos\theta + b$$

Given overall dimensions w and h, and the material dimension, b, what is the value of θ?

An analytical solution for $\theta = f(w, h, b)$ exists, but is not obvious.

Use a numerical root-finding procedure to find the value of θ that satisfies

$$f(\theta) = w \sin\theta - h \cos\theta - b = 0$$

Roots of $f(x) = 0$

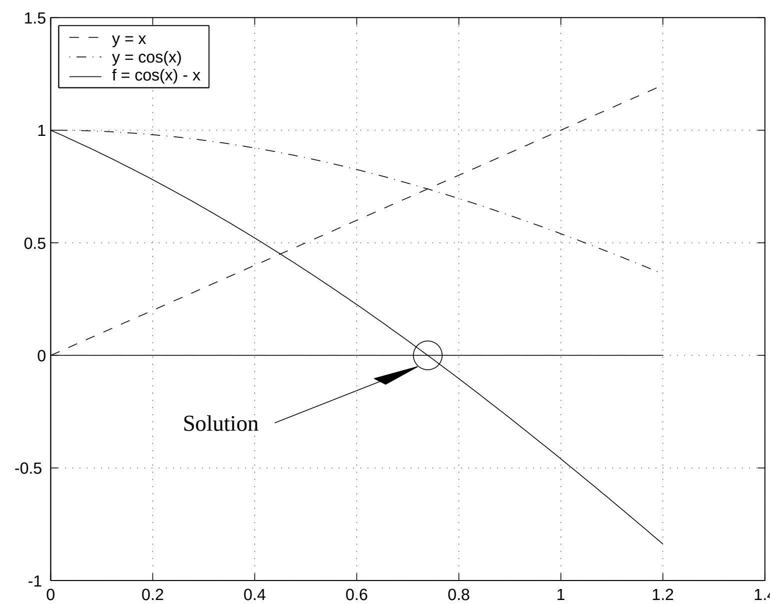
Any function of one variable can be put in the form $f(x) = 0$.

Example: To find the x that satisfies

$$\cos(x) = x,$$

find the zero crossing of

$$f(x) = \cos(x) - x = 0$$



General Considerations

- Is this a special function that will be evaluated often?
- How much precision is needed?
- How fast and robust must the method be?
- Is the function a polynomial?
- Does the function have singularities?

There is no single root-finding method that is best for all situations.

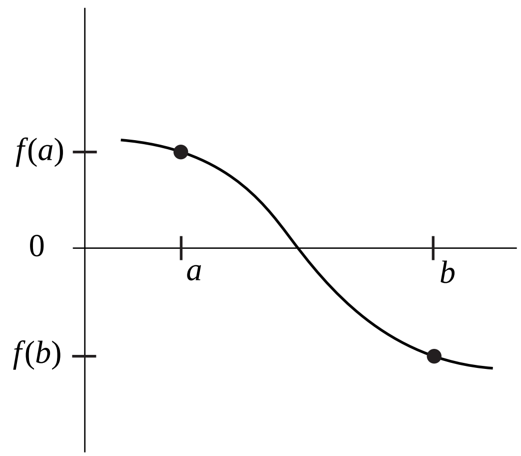
Root-Finding Procedure

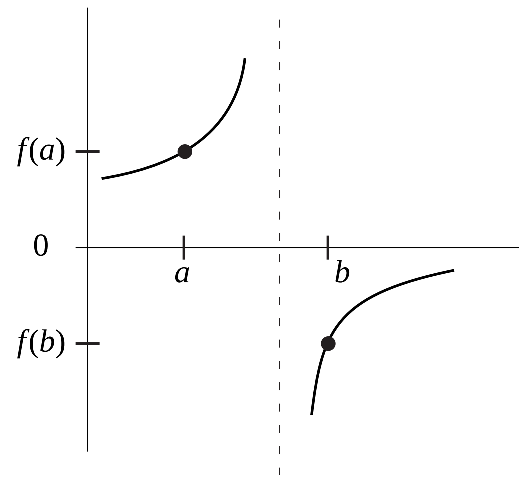
The basic strategy is

1. Plot the function.
   ➢ The plot provides an initial guess, and an indication of potential problems.
2. Select an initial guess.
3. Iteratively refine the initial guess with a root-finding algorithm.

Bracketing

A root is bracketed on the interval $[a, b]$ if $f(a)$ and $f(b)$ have opposite sign. A sign change occurs for singularities as well as roots





Bracketing is used to make initial guesses at the roots, not to accurately estimate the values of the roots.

Bracketing Algorithm (1)

Algorithm 0.1 Bracket Roots

given: $f(x)$, x_{min}, x_{max}, n

$dx = (x_{\text{max}} - x_{\text{min}})/n$
$x_{\text{left}} = x_{\text{min}}$
i = 0

while $i < n$
  $i \leftarrow i + 1$
  $x_{\text{right}} = x_{\text{left}} + dx$
  if $f(x)$ changes sign in $[x_{\text{left}}, x_{\text{right}}]$
    save $[x_{\text{left}}, x_{\text{right}}]$ for further root-finding
  end
  $x_{\text{left}} = x_{\text{right}}$
end

Bracketing Algorithm (2)

A simple test for sign change: $f(a) \times f(b) < 0$?

or in MATLAB

```
if
fa = ...
fb = ...

if fa*fb < 0
    save bracket
end
```

but this test is susceptible to *underflow*.

Bracketing Algorithm (3)

A *better* test uses the built-in `sign` function

```
fa = ...
fb = ...

if sign(fa)~=sign(fb)
    save bracket
end
```

See implementation in the `brackPlot` function

The `brackPlot` Function

`brackPlot` is a NMM toolbox function that

- Looks for brackets of a user-defined $f(x)$
- Plots the brackets and $f(x)$
- Returns brackets in a two-column matrix

Syntax:

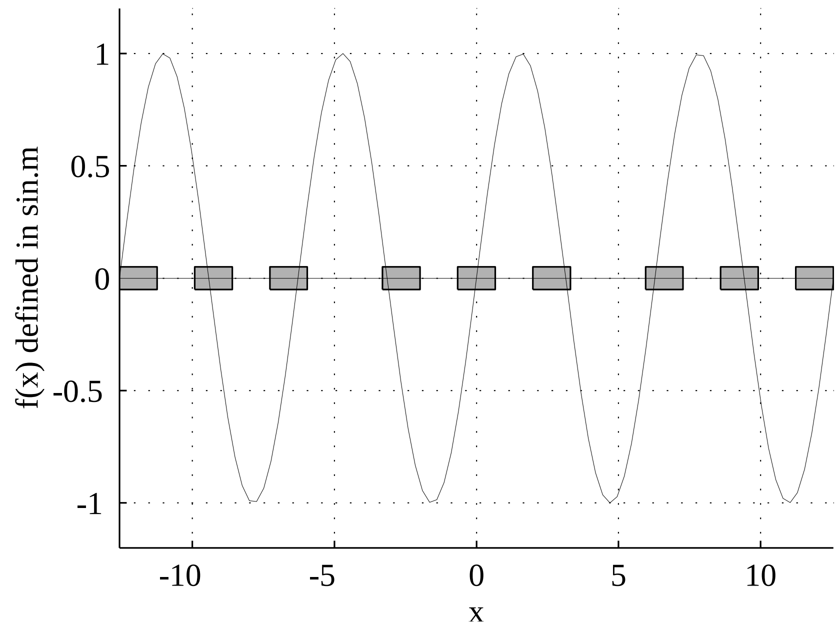
```
brackPlot('myFun',xmin,xmax)
brackPlot('myFun',xmin,xmax,nx)
```

where

myFun	is the name of an m-file that evaluates $f(x)$
xmin, *xmax*	define range of x axis to search
nx	is the number of subintervals on [*xmin*,*xmax*] used to check for sign changes of $f(x)$. Default: *nx*$= 20$

Apply `brackPlot` Function to $\sin(x)$ (1)

```
>> Xb = brackPlot('sin',-4*pi,4*pi)
Xb =
   -12.5664  -11.2436
    -9.9208   -8.5980
    -7.2753   -5.9525
    -3.3069   -1.9842
    -0.6614    0.6614
     1.9842    3.3069
     5.9525    7.2753
     8.5980    9.9208
    11.2436   12.5664
```



Apply `brackPlot` to a user-defined Function (1)

To solve

$$f(x) = x - x^{1/3} - 2 = 0$$

we need an m-file function to evaluate $f(x)$ for any scalar *or* vector of x values.

File `fx3.m`:

Note the use of the array operator.

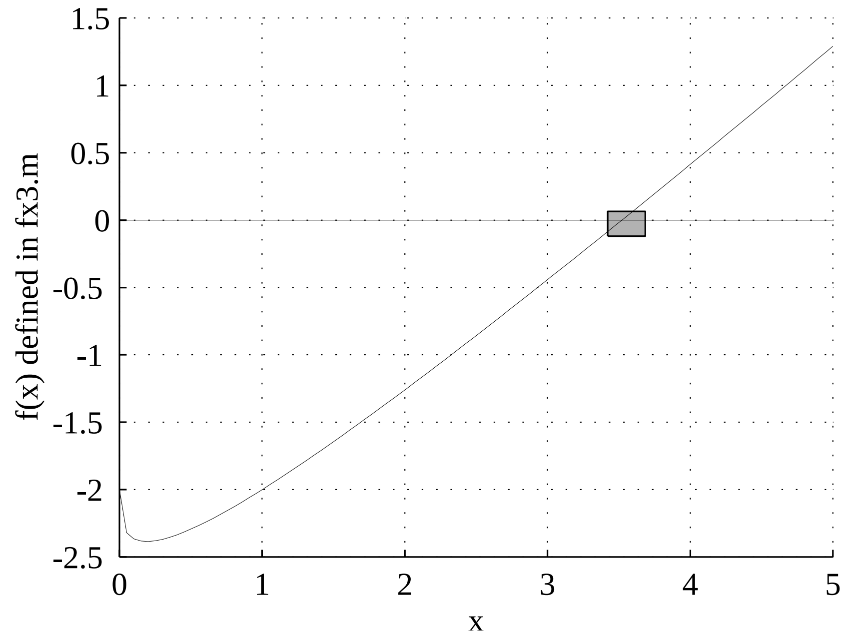
```
function f = fx3(x)
% fx3   Evaluates f(x) = x - x^(1/3) - 2
f = x - x.^(1/3) - 2;
```

Run `brackPlot` with `fx3` as the input function

```
>> brackPlot('fx3',0,5)
ans =
    3.4000    3.6000
```

Apply `brackPlot` to a user-defined Function (2)

```
>> Xb = brackPlot('fx3',0,5)
Xb =
    3.4211    3.6842
```



Apply `brackPlot` to a user-defined Function (3)

Instead of creating a separate m-file, we can use an *anonymous function* object.

```
>> f = @(x) x - x.^(1/3) - 2;
>> f
f =
  function_handle with value:
    @(x)x-x.^(1/3)-2


>> brackPlot(f,0,5)
ans =
    3.4211    3.6842
```

Note: When an anonymous function object is supplied to `brackPlot`, the name of the object is not surrounded in quotes:

`brackPlot(f,0,5)` instead of `brackPlot('fun',0,5)`

Root-Finding Algorithms

We now proceed to develop the following root-finding algorithms:

- Fixed point iteration
- Bisection
- Newton's method
- Secant method

These algorithms are applied *after* initial guesses at the root(s) are identified with bracketing (or guesswork).

Fixed Point Iteration

Fixed point iteration is a simple method. It only works when the iteration function is convergent.

Given $f(x) = 0$, rewrite as $x_{\text{new}} = g(x_{\text{old}})$

Algorithm 0.2 Fixed Point Iteration

initialize: $x_0 = \ldots$
for $k = 1, 2, \ldots$
  $x_k = g(x_{k-1})$
  if converged, stop
end

Convergence Criteria

An automatic root-finding procedure needs to monitor progress toward the root and stop when current guess is close enough to the desired root.

- Convergence checking will avoid searching to unnecessary accuracy.
- Convergence checking can consider whether two successive approximations to the root are close enough to be considered equal.
- Convergence checking can examine whether $f(x)$ is sufficiently close to zero at the current guess.

More on this later . . .

Fixed Point Iteration Example (1)

To solve

$$x - x^{1/3} - 2 = 0$$

rewrite as

$$x_{\rm new} = g_1(x_{\rm old}) = x_{\rm old}^{1/3} + 2$$

or

$$x_{\rm new} = g_2(x_{\rm old}) = \left(x_{\rm old} - 2\right)^3$$

or

$$x_{\rm new} = g_3(x_{\rm old}) = \frac{6 + 2x_{\rm old}^{1/3}}{3 - x_{\rm old}^{2/3}}$$

Are these $g(x)$ functions equally effective?

Fixed Point Iteration Example (2)

$$g_1(x) = x^{1/3} + 2$$

$$g_2(x) = \left(x - 2\right)^3$$

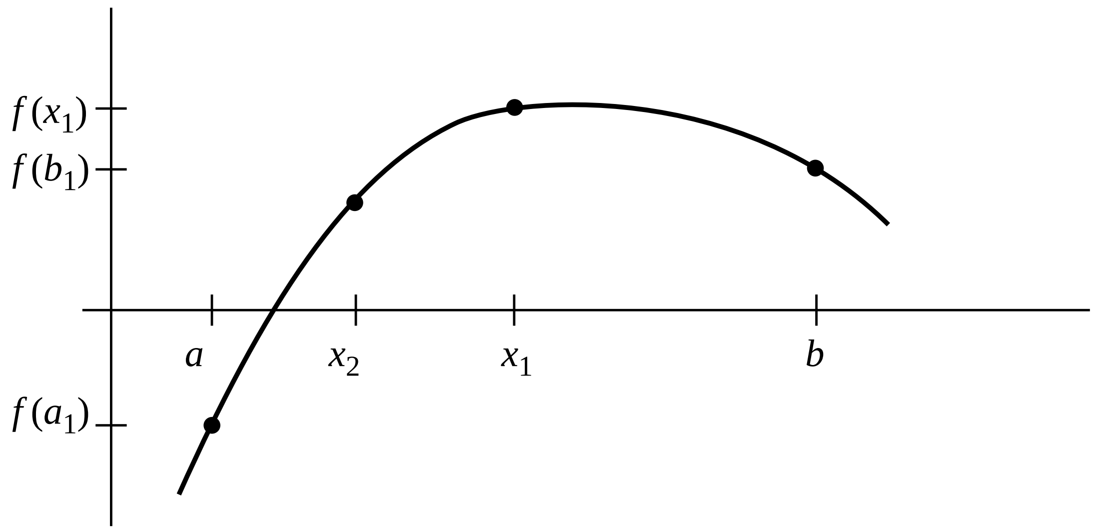
$$g_3(x) = \frac{6 + 2x^{1/3}}{3 - x^{2/3}}$$

k	$g_1(x_{k-1})$	$g_2(x_{k-1})$	$g_3(x_{k-1})$
0	3	3	3
1	3.4422495703	1	3.5266442931
2	3.5098974493	-1	3.5213801474
3	3.5197243050	-27	3.5213797068
4	3.5211412691	-24389	3.5213797068
5	3.5213453678	-1.451×10^{13}	3.5213797068
6	3.5213747615	-3.055×10^{39}	3.5213797068
7	3.5213789946	-2.852×10^{118}	3.5213797068
8	3.5213796042	∞	3.5213797068
9	3.5213796920	∞	3.5213797068

Summary: $g_1(x)$ converges, $g_2(x)$ diverges, $g_3(x)$ converges very quickly

Bisection

Given a bracketed root, halve the interval while continuing to bracket the root



Bisection (2)

For the bracket interval $[a, b]$ the midpoint is

$$x_m = \frac{1}{2}(a + b)$$

A better formula, one that is less susceptible to round-off is

$$x_m = a + \frac{b - a}{2}$$

Bisection Algorithm

Algorithm 0.3 Bisection

initialize: $a = \ldots$, $b = \ldots$
for $k = 1, 2, \ldots$
  $x_m = a + (b - a)/2$
  if $\text{sign}\,(f(x_m)) = \text{sign}\,(f(x_a))$
    $a = x_m$
  else
    $b = x_m$
  end
  if converged, stop
end

Bisection Example

Solve with bisection:

$$x - x^{1/3} - 2 = 0$$

k	a	b	x_{mid}	$f(x_{mid})$
0	3	4		
1	3	4	3.5	-0.01829449
2	3.5	4	3.75	0.19638375
3	3.5	3.75	3.625	0.08884159
4	3.5	3.625	3.5625	0.03522131
5	3.5	3.5625	3.53125	0.00845016
6	3.5	3.53125	3.515625	-0.00492550
7	3.51625	3.53125	3.5234375	0.00176150
8	3.51625	3.5234375	3.51953125	-0.00158221
9	3.51953125	3.5234375	3.52148438	0.00008959
10	3.51953125	3.52148438	3.52050781	-0.00074632

Analysis of Bisection (1)

Let δ_n be the size of the bracketing interval at the n^{th} stage of bisection. Then

$$\delta_0 = b - a = \text{initial bracketing interval}$$

$$\delta_1 = \frac{1}{2}\delta_0$$

$$\delta_2 = \frac{1}{2}\delta_1 = \frac{1}{4}\delta_0$$

$$\vdots$$

$$\delta_n = \left(\frac{1}{2}\right)^n \delta_0$$

$$\Longrightarrow \qquad \frac{\delta_n}{\delta_0} = \left(\frac{1}{2}\right)^n = 2^{-n}$$

or

$$n = \log_2\left(\frac{\delta_n}{\delta_0}\right)$$

Analysis of Bisection (2)

$$\frac{\delta_n}{\delta_0} = \left(\frac{1}{2}\right)^n = 2^{-n} \qquad \text{or} \qquad n = \log_2\left(\frac{\delta_n}{\delta_0}\right)$$

n	$\dfrac{\delta_n}{\delta_0}$	function evaluations
5	3.1×10^{-2}	7
10	9.8×10^{-4}	12
20	9.5×10^{-7}	22
30	9.3×10^{-10}	32
40	9.1×10^{-13}	42
50	8.9×10^{-16}	52

Convergence Criteria

An automatic root-finding procedure needs to monitor progress toward the root and stop when current guess is close enough to the desired root.

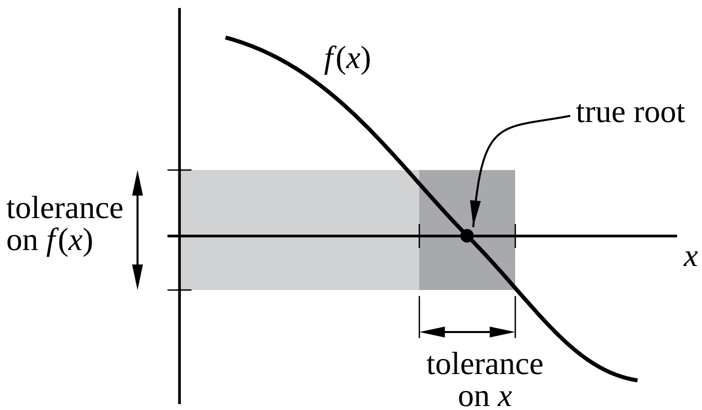
- Convergence checking will avoid searching to unnecessary accuracy.
- Check whether successive approximations are close enough to be considered the same:

$$|x_k - x_{k-1}| < \delta_x$$

- Check whether $f(x)$ is close enough zero.

$$|f(x_k)| < \delta_f$$

Convergence Criteria on x



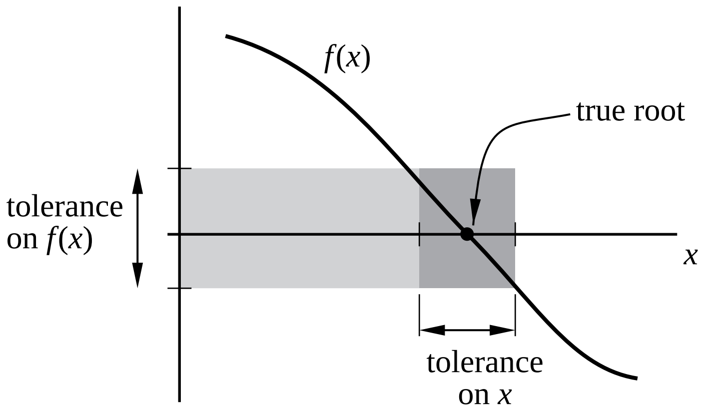
Absolute tolerance: $\left|x_k - x_{k-1}\right| < \delta_x$

Relative tolerance: $\left|\dfrac{x_k - x_{k-1}}{b - a}\right| < \hat{\delta}_x$

x_k = current guess at the root

x_{k-1} = previous guess at the root

Convergence Criteria on $f(x)$



Absolute tolerance: $\left|f(x_k)\right| < \delta_f$

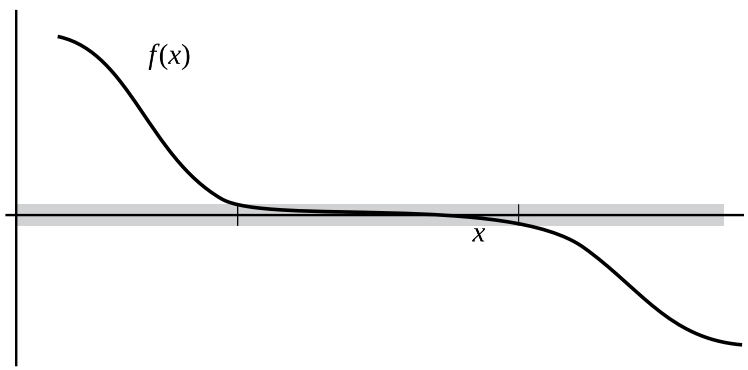
Relative tolerance:

$$|f(x_k)| < \hat{\delta}_f \max\Big\{|f(a_0)|, |f(b_0)|\Big\}$$

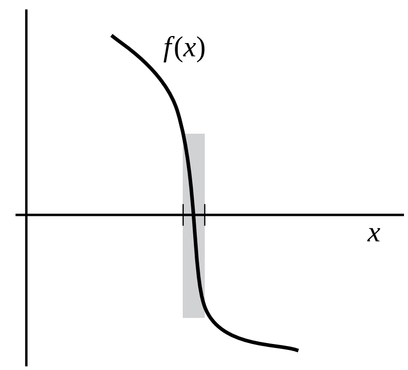
where a_0 and b_0 are the original brackets

Convergence Criteria on $f(x)$

If $f'(x)$ is small near the root, it is easy to satisfy a tolerance on $f(x)$ for a large range of Δx. A tolerance on Δx is more conservative.

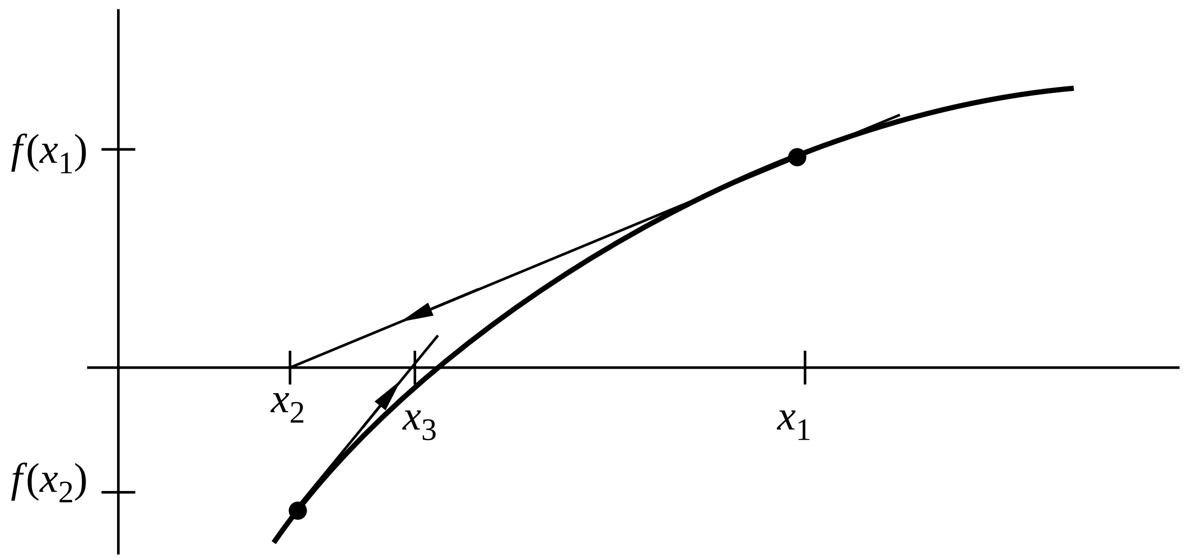


If $f'(x)$ is large near the root, it is possible to satisfy a tolerance on Δx when $|f(x)|$ is still large. A tolerance on $f(x)$ is more conservative.



Newton's Method (1)

For a current guess x_k, use $f(x_k)$ and the slope $f'(x_k)$ to predict where $f(x)$ crosses the x axis.



Newton's Method (2)

Expand $f(x)$ in Taylor Series around x_k

$$f(x_k + \Delta x) = f(x_k) + \Delta x \left.\frac{df}{dx}\right|_{x_k} + \frac{(\Delta x)^2}{2} \left.\frac{d^2 f}{dx^2}\right|_{x_k} + \ldots$$

Substitute $\Delta x = x_{k+1} - x_k$ and neglect second order terms to get

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

where

$$f'(x_k) = \left.\frac{df}{dx}\right|_{x_k}$$

Newton's Method (3)

Goal is to find x such that $f(x) = 0$.

Set $f(x_{k+1}) = 0$ and solve for x_{k+1}

$$0 = f(x_k) + (x_{k+1} - x_k)\, f'(x_k)$$

or, solving for x_{k+1}

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

Newton's Method Algorithm

Algorithm 0.4

> initialize: $x_1 = \ldots$
> for $k = 2, 3, \ldots$
> $x_k = x_{k-1} - f(x_{k-1})/f'(x_{k-1})$
>   if converged, stop
> end

Newton's Method Example (1)

Solve:

$$x - x^{1/3} - 2 = 0$$

First derivative is

$$f'(x) = 1 - \frac{1}{3}x^{-2/3}$$

The iteration formula is

$$x_{k+1} = x_k - \frac{x_k - x_k^{1/3} - 2}{1 - \frac{1}{3}x_k^{-2/3}}$$

Newton's Method Example (2)

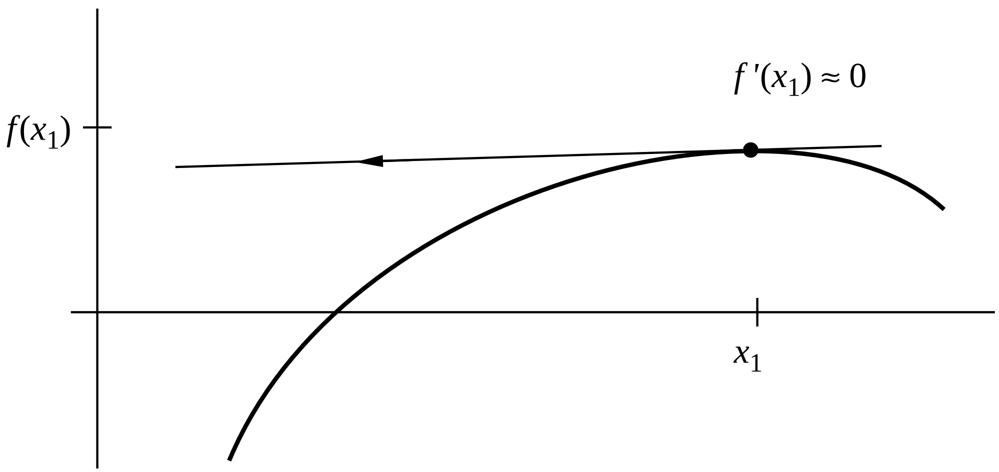
$$x_{k+1} = x_k - \frac{x_k - x_k^{1/3} - 2}{1 - \frac{1}{3}x_k^{-2/3}}$$

k	x_k	$f'(x_k)$	$f(x)$
0	3	0.83975005	-0.44224957
1	3.52664429	0.85612976	0.00450679
2	3.52138015	0.85598641	3.771×10^{-7}
3	3.52137971	0.85598640	2.664×10^{-15}
4	3.52137971	0.85598640	0.0

Conclusion

- Newton's method converges *much* more quickly than bisection
- Newton's method requires an analytical formula for $f'(x)$
- The algorithm is simple as long as $f'(x)$ is available.
- Iterations are not guaranteed to stay inside an ordinal bracket.

Divergence of Newton's Method



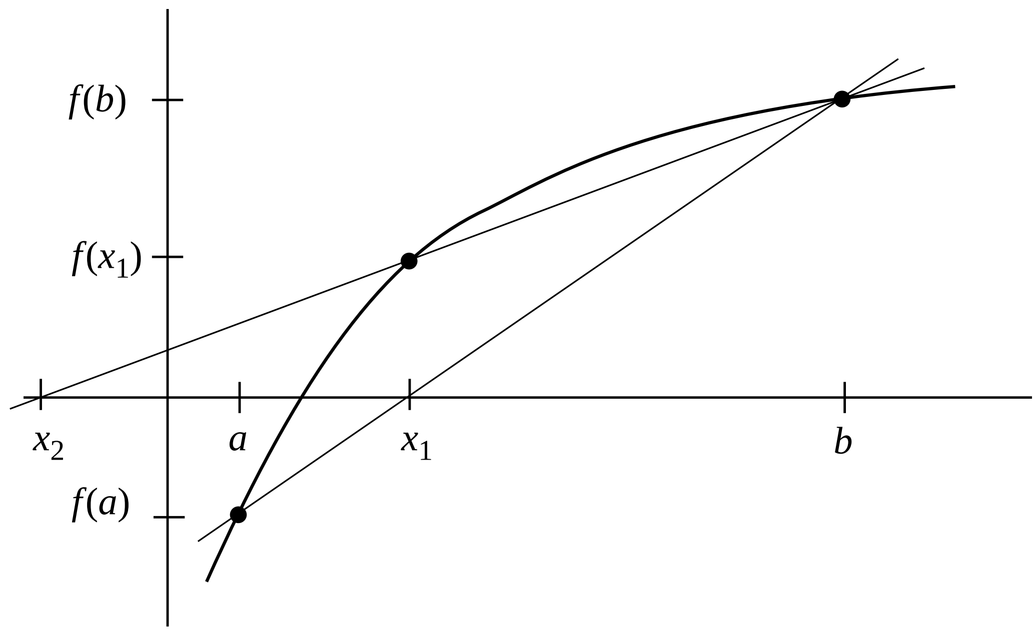
Since

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

the new guess, x_{k+1}, will be far from the old guess whenever $f'(x_k) \approx 0$

Secant Method (1)

Given two guesses x_{k-1} and x_k, the next guess at the root is where the line through $f(x_{k-1})$ and $f(x_k)$ crosses the x axis.



Secant Method (2)

Given

$$x_k = \text{current guess at the root}$$
$$x_{k-1} = \text{previous guess at the root}$$

Approximate the first derivative with

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

Substitute approximate $f'(x_k)$ into formula for Newton's method

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

to get

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

Secant Method (3)

Two versions of this formula are equivalent in exact math:

$$x_{k+1} = x_k - f(x_k)\left[\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}\right] \qquad (\star)$$

and

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

Equation ($\star$) is better since it is of the form $x_{k+1} = x_k + \Delta$. Even if Δ is inaccurate the change in the estimate of the root will be small at convergence because $f(x_k)$ will also be small.

Equation ($\star\star$) is susceptible to catastrophic cancellation:

- $f(x_k) \to f(x_{k-1})$ as convergence approaches, so cancellation error in the denominator can be large.
- $|f(x)| \to 0$ as convergence approaches, so underflow is possible

Secant Algorithm

Algorithm 0.5

initialize: $x_1 = \ldots$, $x_2 = \ldots$
for $k = 2, 3 \ldots$
  $x_{k+1} = x_k$
    $-f(x_k)(x_k - x_{k-1})/(f(x_k) - f(x_{k-1}))$
  if converged, stop
end

Secant Method Example

Solve:

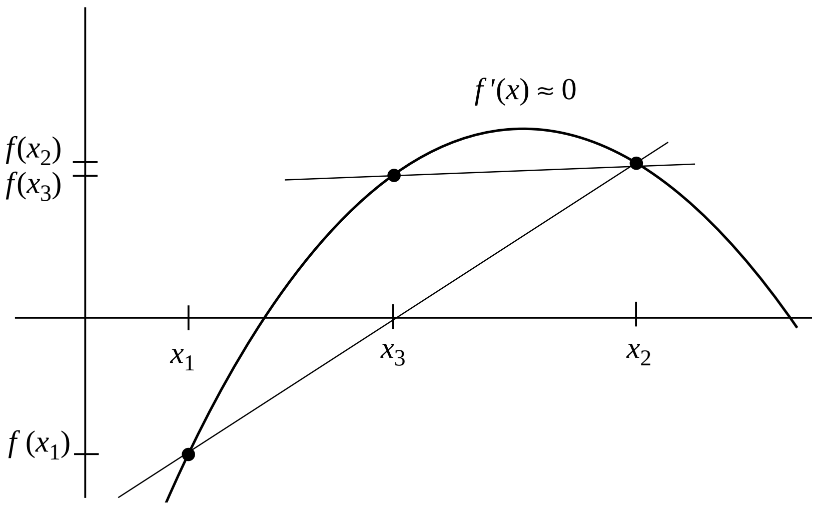
$$x - x^{1/3} - 2 = 0$$

k	x_{k-1}	x_k	$f(x_k)$
0	4	3	-0.44224957
1	3	3.51734262	-0.00345547
2	3.51734262	3.52141665	0.00003163
3	3.52141665	3.52137970	-2.034×10^{-9}
4	3.52137959	3.52137971	-1.332×10^{-15}
5	3.52137971	3.52137971	0.0

Conclusions

- Converges almost as quickly as Newton's method.
- No need to compute $f'(x)$.
- The algorithm is simple.
- Two initial guesses are necessary
- Iterations are not guaranteed to stay inside an ordinal bracket.

Divergence of Secant Method



Since

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

the new guess, x_{k+1}, will be far from the old guess whenever $f'(x_k) \approx f(x_{k-1})$ and $|f(x)|$ is not small.

Summary of Basic Root-finding Methods

- Plot $f(x)$ before searching for roots
- Bracketing finds coarse interval containing roots and singularities
- Bisection is robust, but converges slowly
- Newton's Method
  - ▷ Requires $f(x)$ and $f'(x)$.
  - ▷ Iterates are not confined to initial bracket.
  - ▷ Converges rapidly.
  - ▷ Diverges if $f'(x) \approx 0$ is encountered.
- Secant Method
  - ▷ Uses $f(x)$ values to approximate $f'(x)$.
  - ▷ Iterates are not confined to initial bracket.
  - ▷ Converges almost as rapidly as Newton's method.
  - ▷ Diverges if $f'(x) \approx 0$ is encountered.

`fzero` Function (1)

`fzero` is a hybrid method that combines bisection, secant and reverse quadratic interpolation

Syntax:

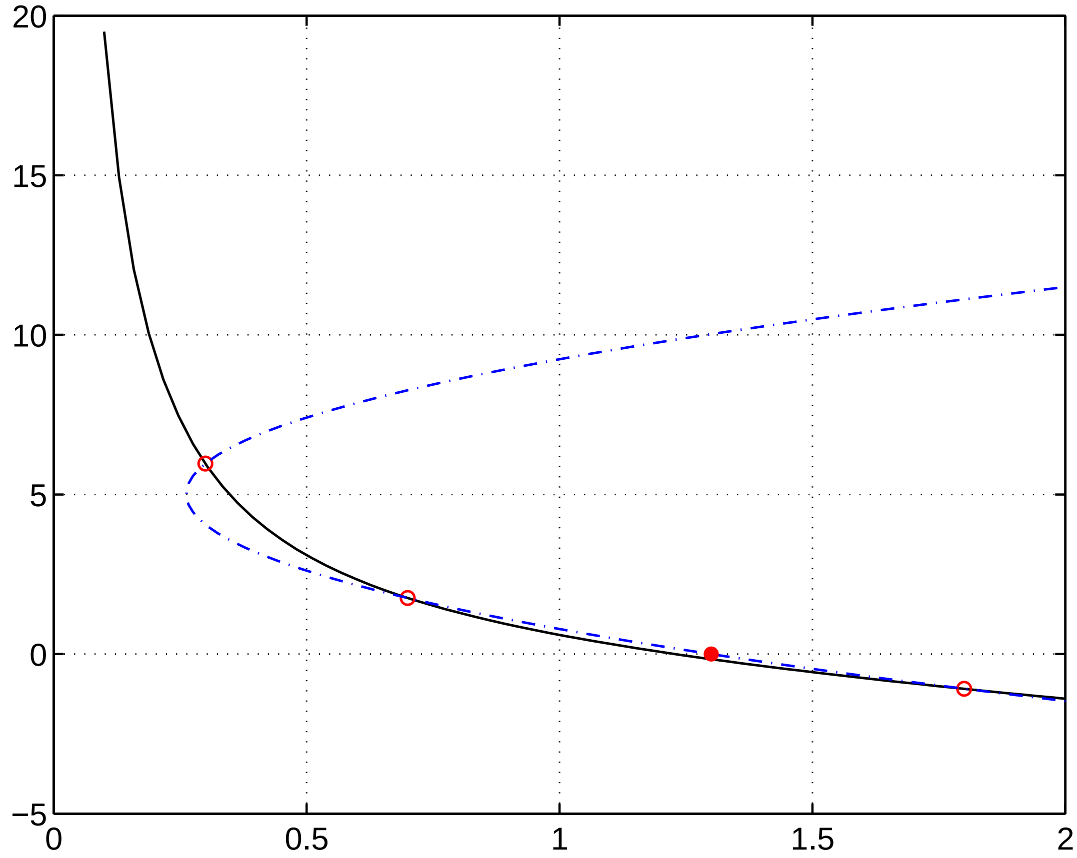
```
r = fzero('fun',x0)
r = fzero('fun',x0,options)
```

`x0` can be a scalar or a two element vector

- If `x0` is a scalar, `fzero` tries to create its own bracket.
- If `x0` is a two element vector, `fzero` uses the vector as a bracket.

Reverse Quadratic Interpolation

Find the point where the x axis intersects the sideways parabola passing through three pairs of $(x, f(x))$ values.



`fzero` Function (2)

`fzero` chooses next root as

- Result of reverse quadratic interpolation (RQI) if that result is inside the current bracket.
- Result of secant step if RQI fails, and if the result of secant method is in inside the current bracket.
- Result of bisection step if both RQI and secant method fail to produce guesses inside the current bracket.

fzero Function (3)

Optional parameters to control fzero are specified with the optimset function.

Examples:
Tell fzero to display the results of each step:

```
>> options = optimset('Display','iter');
>> x = fzero('myFun',x0,options)
```

Tell fzero to use a relative tolerance of 5×10^{-9}:

```
>> options = optimset('TolX',5e-9);
>> x = fzero('myFun',x0,options)
```

Tell fzero to suppress all printed output, and use a relative tolerance of 5×10^{-4}:

```
>> options = optimset('Display','off','TolX',5e-4);
>> x = fzero('myFun',x0,options)
```

`fzero` Function (4)

Allowable options (specified via `optimset`):

Option type	Value	Effect
`'Display'`	`'iter'`	Show results of each iteration
	`'final'`	Show root and original bracket
	`'off'`	Suppress all print out
`'TolX'`	`tol`	Iterate until $$\lvert\Delta x\rvert < \max\left[\texttt{tol}, \texttt{tol} * \texttt{a}, \texttt{tol} * \texttt{b}\right]$$ where $\Delta x = (b-a)/2$, and $[a, b]$ is the current bracket.

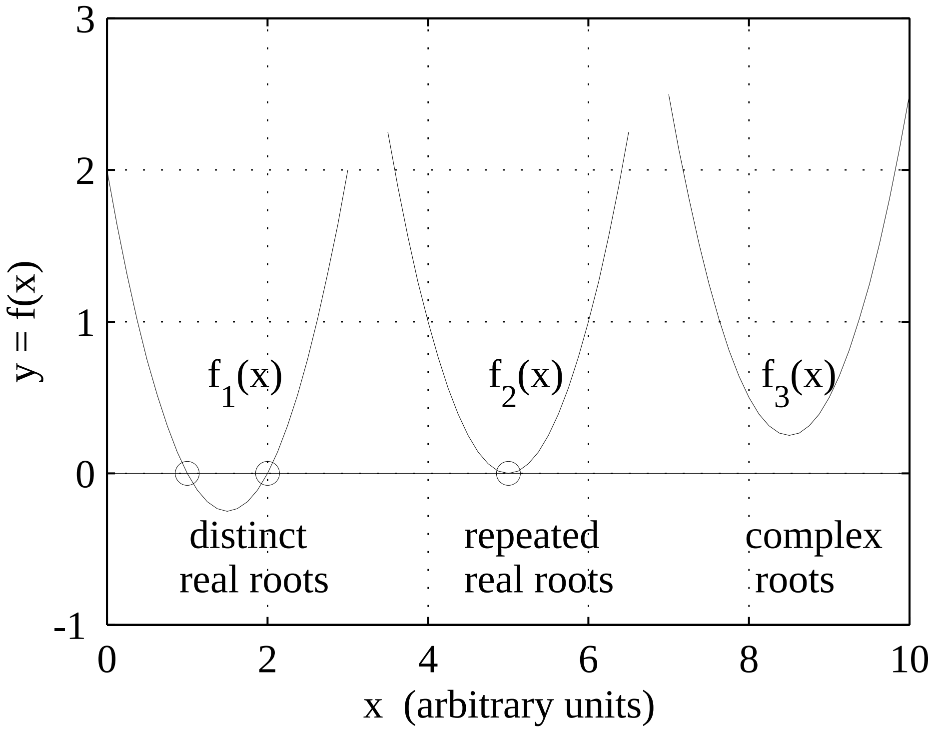
The default values of `'Display'` and `'TolX'` are equivalent to

```
options = optimset('Display','iter','TolX',eps)
```

Roots of Polynomials

Complications arise due to

- Repeated roots
- Complex roots
- Sensitivity of roots to small perturbations in the polynomial coefficients (conditioning).



Algorithms for Finding Polynomial Roots

- Bairstow's method
- Müller's method
- Laguerre's method
- Jenkin's–Traub method
- Companion matrix method

`roots` **Function (1)**

The built-in `roots` function uses the companion matrix method

- No initial guess
- Returns *all* roots of the polynomial
- Solves eigenvalue problem for companion matrix

Write polynomial in the form

$$c_1 x^n + c_2 x^{n-1} + \ldots + c_n x + c_{n+1} = 0$$

Then, for a *third* order polynomial

```
>> c = [c1 c2 c3 c4];
>> r = roots(c)
```

`roots` Function (2)

The eigenvalues of

$$A = \begin{bmatrix} -c_2/c_1 & -c_3/c_1 & -c_4/c_1 & -c_5/c_1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

are the same as the roots of

$$c_5\lambda^4 + c_4\lambda^3 + c_3\lambda^2 + c_2\lambda + c_1 = 0.$$

`roots` Function (3)

The statements

```
c = ...       %  vector of polynomial coefficients
r = roots(c);
```

are equivalent to

```
c = ...
n = length(c);
A = diag(ones(1,n-2),-1);   %  ones on first subdiagonal
A(1,:) = -c(2:n) ./ c(1);   %  first row is -c(j)/c(1), j=2..n
r = eig(A);
```

`roots` Examples

Roots of

$$f_1(x) = x^2 - 3x + 2$$

$$f_2(x) = x^2 - 10x + 25$$

$$f_3(x) = x^2 - 17x + 72.5$$

are found with

```
>> roots([1 -3 2])
ans =
        2
        1

>> roots([1 -10 25])
ans =
        5
        5

>> roots([1 -17 72.5])
ans =
    8.5000 + 0.5000i
    8.5000 - 0.5000i
```