module 03
solving $f(x) = 0$

ahmad taha — spring 2026

ce 2989 — numerical methods in cee

email: ahmad.taha@vanderbilt.edu

webpage: http://lab.vanderbilt.edu/taha

VANDERBILT UNIVERSITY

February 11, 2026

solving nonlinear system of equations: cee applications

- **single variable systems** ($x \in \mathbb{R}$)
  - *open channel flow (manning's eq):* solve for normal depth y_n

$$f(y_n) = \frac{1}{n} a(y_n)[r(y_n)]^{2/3} s^{1/2} - q = 0$$

  - *pipe friction (colebrook-white):* solve for friction factor f

$$\frac{1}{\sqrt{f}} + 2\log\left(\frac{\epsilon}{3.7d} + \frac{2.51}{re\sqrt{f}}\right) = 0$$

- **multivariable systems** ($x \in \mathbb{R}^n$): in large-scale networked infrastructure
  - *traffic assignment:* user equilibrium flows x_a

$$t_a(x_a) = t_0\left[1 + \alpha\left(\frac{x_a}{c}\right)^{\beta}\right]$$

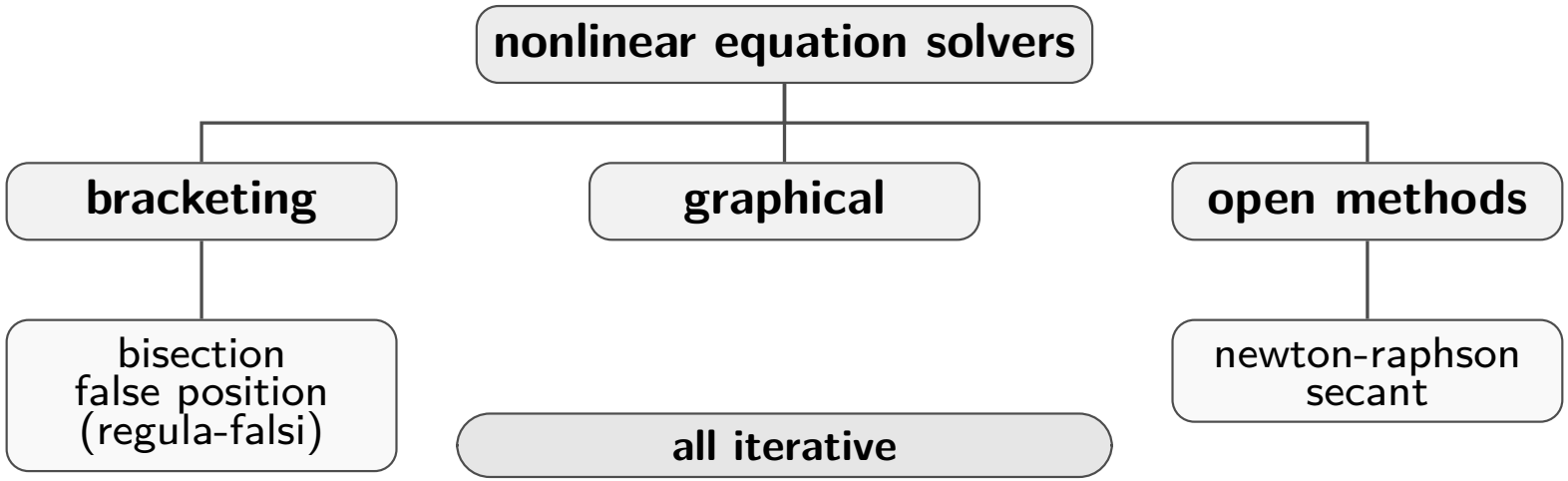
  - *gps positioning:* solve for coordinates (x, y, z) and bias δt

$$\rho_i - \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - c\delta t = 0$$

- in this class, we will mostly focus on single dimension or $x \in \mathbb{R}$ to solve $f(x) = 0$

module outline: solving nl equation $f(x) = 0$ for x

- bisection method
- false-position method
- fixed point iteration
- newton-raphson method
- secant method
- brent's method
- extensions to higher dimensions



module objective

- we will learn a collection of methods to solve $f(x) = 0$ for a single variable $x \in \mathbb{R}$ and a function $f(x) : \mathbb{R} \to \mathbb{R}$
- objective isn't *just* to learn how to follow an algorithm and its steps
- but to learn how to analyze error convergence and properties of each algorithm
- so you can pick the correct method/algorithm for your specific problem
- **example:** we know the roots of

$$f(x) = ax^2 + bx + c = 0 \quad \to \quad x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- what if we need to find roots for

$$f(x) = x^2 + \sin(x) + 10 = 0 \quad \text{or} \quad f(x) = x^7 + 7x^4 - \pi = 0$$

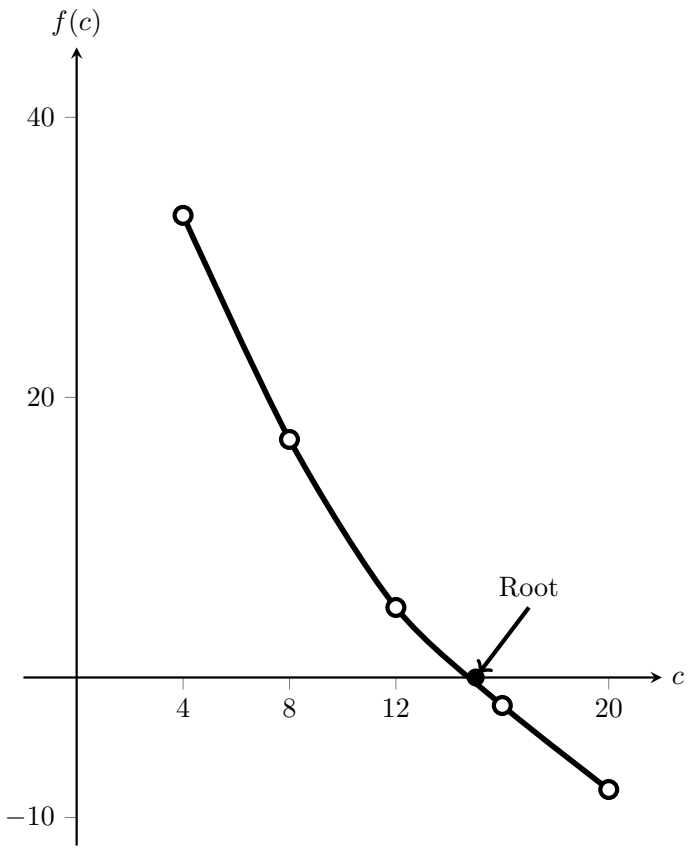
- majority of nl (nonlinear) equations in the world have **no explicit or closed form solution** as the quadratic polynomial
- do all polynomials have closed form expressions like quadratics?
- hence, we need to resort to numerical methods to find roots
- nl equations model: energy balance, mass balance, conservation of energy/momentum/etc..

example and outline of the methods

- recall the example where we obtained an analytical solution of the ODE as

$$v(t) = \frac{gm}{c}(1 - e^{-c/m \cdot t})$$

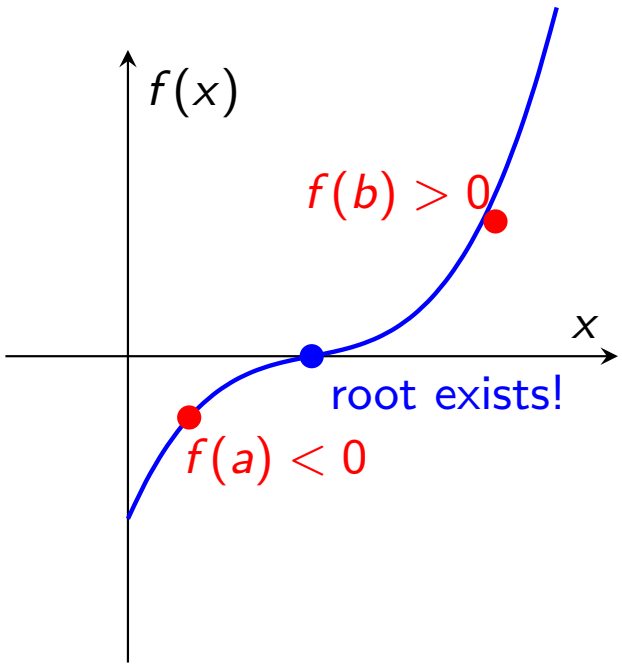
- suppose you want to compute the drag coefficient c so that a parachutist of mass m has a particular velocity at time t
- can express this problem as $f(c) = 0$
- graphical solution:



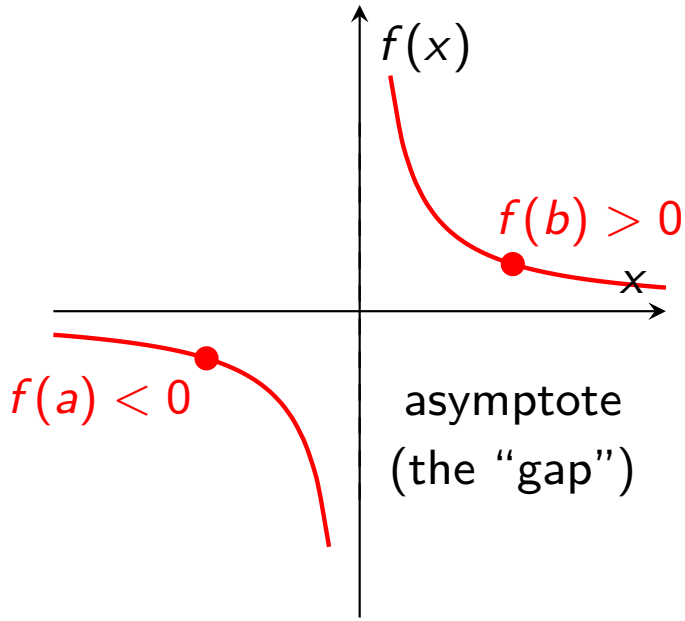
what are the conditions?

- is there always a solution to $f(x) = 0$?
- what are the needed conditions? continuity (no teleportation) + sign change (crossing over)
- example:

success:

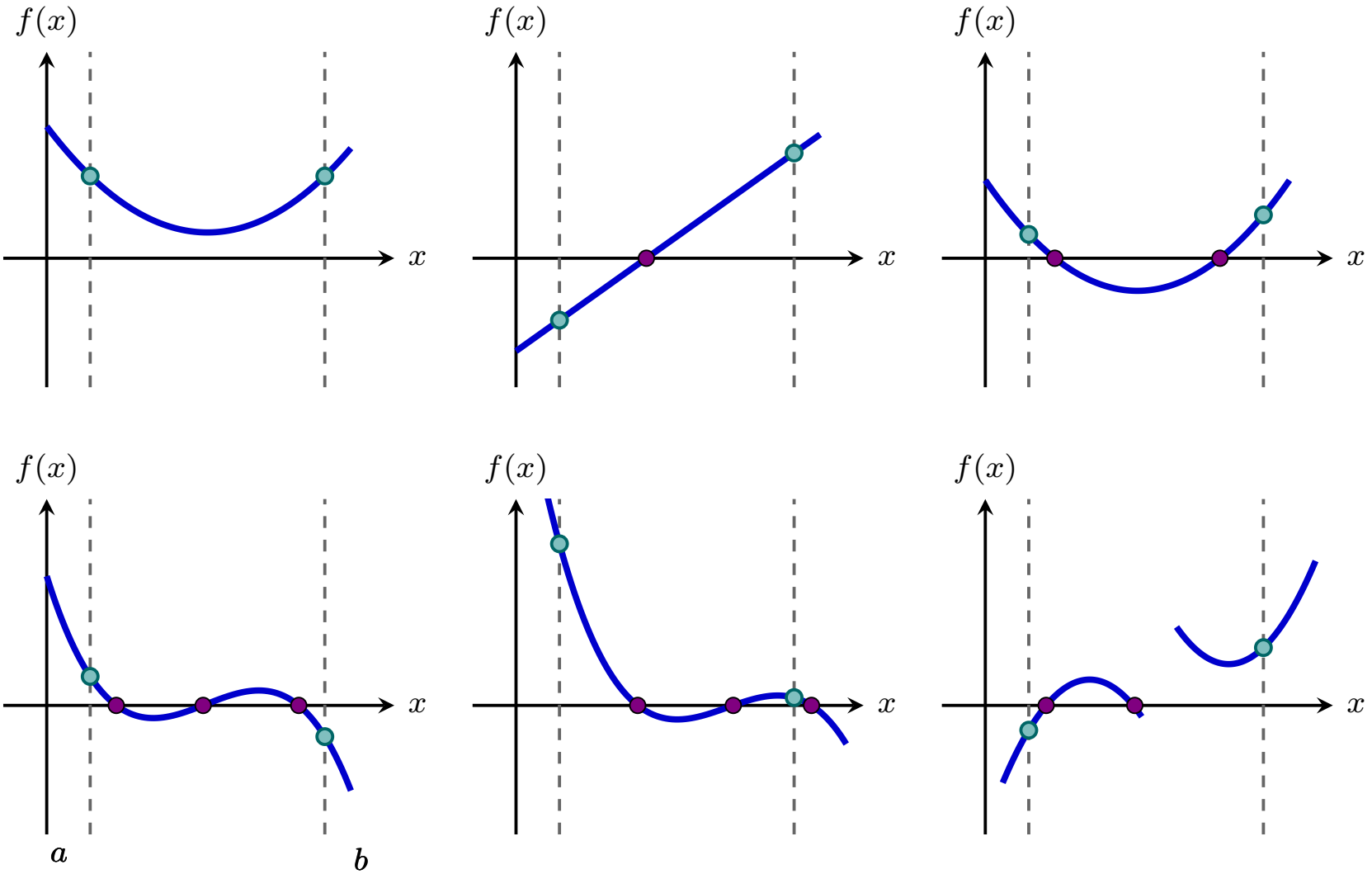


failure:



possible ways solutions exist to $f(x) = 0$

- roots of $f(x)$ for $x \in [a, b]$ could have multiple cases



bracketing methods

- bracketing methods rely on one fundamental idea
- notice how the function $f(x)$ changes signs before and after the root
- for bracketing methods, two initial guesses for the root are needed
- they're not really guesses but more or less boundary points of where the engineer thinks that the roots will exist within
- that is: $x \in [a, b]$ lower and upper bounds defining the bracket
- useful result: if one root x^* of a real and continuous function is bounded by values $x = a$ and $x = b$ then if

$$f(a)f(b) < 0$$

  the function changes sign on opposite side of the root
- bracketing methods: iteratively reduce the bracket size $b - a$ until the bracket size is so tiny (while still $f(a)f(b) < 0$) so that we know the solution lies within $[a, b]$
- how to reduce bracket size? divide by half? more? less?

bracketing method 1: the bisection method

- bisection method (also called binary chopping, interval halving, or bolzano's method): interval $[a, b]$ is always divided by half
- we choose the bracket so function changes sign over an interval $f(a)f(b) < 0$ meaning that the root x^* is definitely between a and b
  1. compute function value at midpoint $x = 0.5(a + b)$
  2. if at x, $f(a)f(x) < 0 \Rightarrow$ root x^* is between $[a, x]$ then set $b = x$ and re-iterate
  3. otherwise $f(a)f(x) > 0$, root x^* is between $[x, b]$ and set $a = x$ and re-iterate
  4. if $f(x) = 0$ then set $x^* = x$ and terminate algorithm
- how to terminate? well you could use any termination criteria
  - absolute error tolerance
  - number of sig figs
  - orrrrr relative approximation error $\epsilon_{ar} = \text{are}$ which is what we will be using

bisection method – matlab code

```
function [root,fx,ea,iter]=bisect(func,xl,xu,es,maxit,varargin)
% xl, xu = lower and upper guesses
% es = desired relative error (default = 0.0001%)
% maxit = maximum allowable iterations (default = 50)
% p1,p2,... = additional parameters used by func
% output:
% root = real root
% fx = function value at root
% ea = approximate relative error (%)
if nargin<3,error('at least 3 input arguments required'),end
test = func(xl,varargin{:})*func(xu,varargin{:});
if test>0,error('no sign change'),end
if nargin<4||isempty(es), es=0.0001;end
if nargin<5||isempty(maxit), maxit=50;end
iter = 0; xr = xl; ea = 100;
while (1)
xrold = xr;
xr = (xl + xu)/2;
iter = iter + 1;
if xr ~= 0,ea = abs((xr - xrold)/xr) * 100;end
test = func(xl,varargin{:})*func(xr,varargin{:});
if test < 0
xu = xr;
elseif test > 0
xl = xr;
else
ea = 0;
end
if ea <= es || iter >= maxit,break,end
end
root = xr; fx = func(xr, varargin{:});
```

pros + cons of bisection

pros:

- easy
- can always find a root (assuming you start in a bracket that contains a root)
- number of iterations required to attain a specific absolute error can be computed a priori

cons:

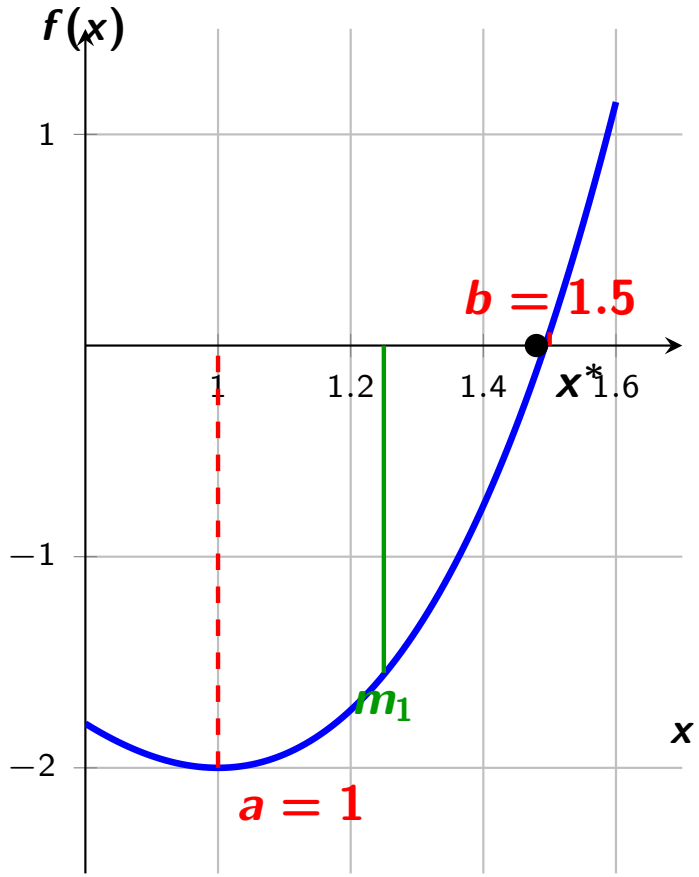
- super slow + need to know bounds for the roots (a and b)
- doesn't address the issue of multiple roots
- no account is taken for the magnitude of $f(a)$ and $f(b)$: if $f(a)$ is closer to zero, it is likely that root is closer to a so iterating as in bisection wastes so much time/iterations

bisection method example

- consider the polynomial $f(x) = x^4 - 4x + 1$ with initial bracket $[a, b] = [1, 1.5]$
- sign check: $f(1) = -2 < 0$, $f(1.5) = 0.0625 > 0$ so a root lies in $[1, 1.5]$ which means we can proceed
- bisection method updates (midpoint $m = (a + b)/2$):

iter	m	new bracket
1	1.25	[1.25, 1.5]
2	1.375	[1.375, 1.5]
3	1.4375	[1.4375, 1.5]
4	1.46875	[1.46875, 1.5]
5	1.484375	[1.46875, 1.484375]
6	1.4765625	[1.4765625, 1.484375]
7	1.48046875	[1.48046875, 1.484375]
8	1.482421875	[1.48046875, 1.482421875]

- final bracket: $[1.4804, 1.4824]$
- root: $\boxed{x \approx 1.48}$



error analysis

- we can determine number of iterations needed for the bisection method, given a certain absolute error
- how? well the bracket length at first iteration is $b_1 - a_1 = b - a$
- bracket length in the second iteration: $b_2 - a_2 = 1/2(b_1 - a_1)$, 3rd iteration: $b_3 - a_3 = 1/4(b_1 - a_1)$, etc..
- this means that at every iteration, the bracket length is divided by two, so after n iterations, the bracket length will be divided by 2^n
- this means that the distance to the actual solution is upper bounded
- then we can write for $n \geq 1$, $b_n - a_n = \dfrac{1}{2^{n-1}}(b - a)$

$$x_n = \frac{1}{2}(a_n + b_n) \quad \rightarrow \boxed{|x^* - x_n| \leq \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b - a)}$$

- **example:** compute number of iterations needed to solve $x^3 + 4x^2 - 10 = 0$ with accuracy 10^{-3} (absolute error) using $[a, b] = [1, 2]$
- **solution:** need absolute error at iteration n to be at most 10^{-3} hence

$$|x^* - x_n| \leq \frac{2 - 1}{2^n} \leq 10^{-3}$$

which implies that we need $2^n \geq 10^3$ or $n \geq \dfrac{3\ln(10)}{\ln(2)} = 9.96$ so $n = 10$

another example

- consider this interval for the bisection method: $[45, 60]$
- compute the number of steps to compute a root with a relative error less than or equal 10^{-8}
- **solution:** we'll use the previous result
- the solution x^* should satisfy

$$|x^* - x_n| \leq \frac{60 - 45}{2^n}$$

- but the problem asks for the relative error

$$\frac{|x^* - x_n|}{x^*} \leq \frac{15}{x^* 2^n} \leq 10^{-8}$$

- this can be simplified as

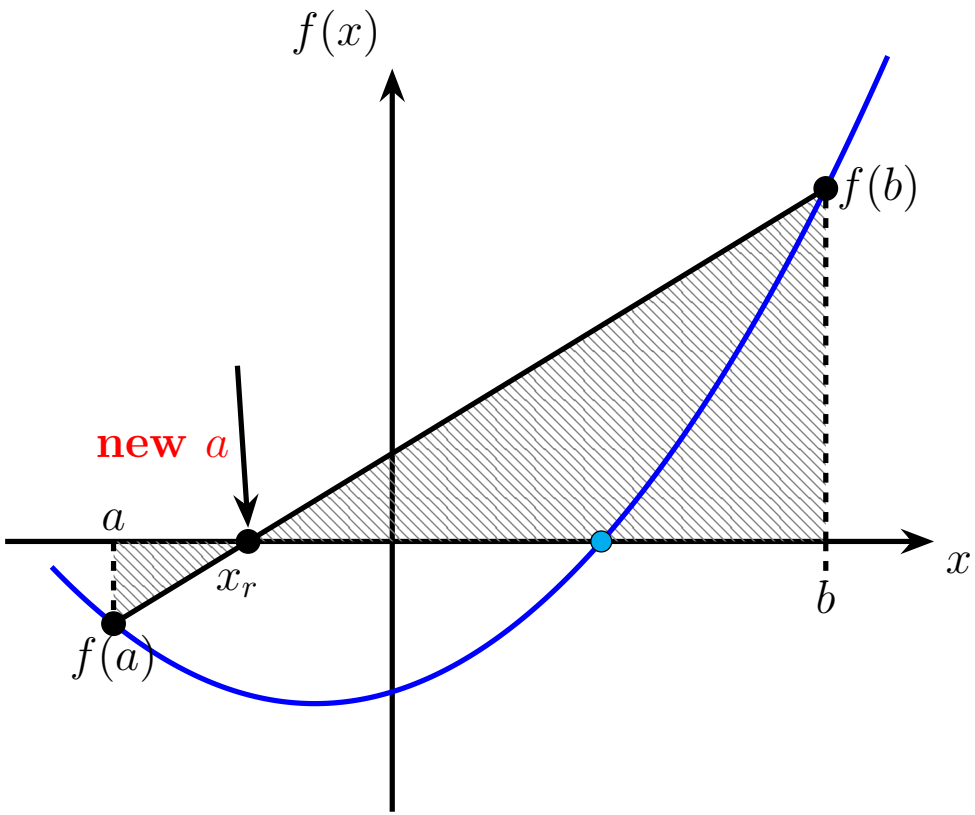
$$\frac{15}{x^* 2^n} \leq 10^{-8}$$

- but the value of x^* will be in the interval $[45, 60]$
- this can be written as

$$\max_{x^* \in [45,60]} \frac{15}{x^* 2^n} \leq 10^{-8} \quad \Rightarrow \quad \frac{1}{3 \times 2^n} \leq 10^{-8}$$

so $n = 25$

false position method



- real root is bounded by a and b of f(x) $= 0$
- we can approximate the solution by drawing a linear line $l(x)$ between the points $[a, f(a)]$ and $[b, f(b)]$
- then we find x^* such that $l(x^*) = 0$
- notice how the two shaded triangles are similar triangles
- this means that

$$\frac{f(a)}{x_r - a} = \frac{f(b)}{x_r - b}$$

- this means that we can compute x_r via this equation

$$x_r = b - \frac{f(b)(a-b)}{f(a) - f(b)}$$

  - if $f(x_r) < 0$ then $a = x_r$
  - if $f(x_r) > 0$ then $b = x_r$

comparing false position and bisection

- usually false position performs better than bisection
- but not always
- **example:** compare bisection and false position methods to compute the room of $f(x) = x^{10} - 1$ for $x \in [0, 1.3]$

bisection:

Iteration	a	b	x_r	Absolute Error (%)	Relative Error (%)
1	0	1.3	0.65	100.0	35
2	0.65	1.3	0.975	33.3	2.5
3	0.975	1.3	1.1375	14.3	13.8
4	0.975	1.1375	1.05625	7.7	5.6
5	0.975	1.05625	1.015625	4.0	1.6

false position

Iteration	a	b	x_r	Absolute Error (%)	Relative Error (%)
1	0	1.3	0.09430	–	90.6
2	0.09430	1.3	0.18176	48.1	81.8
3	0.18176	1.3	0.26287	30.9	73.7
4	0.26287	1.3	0.33811	22.3	66.2
5	0.33811	1.3	0.40788	17.1	59.2

- fixing the issues of false position? possible to modify it to fix the issue

open methods and fixed point iteration

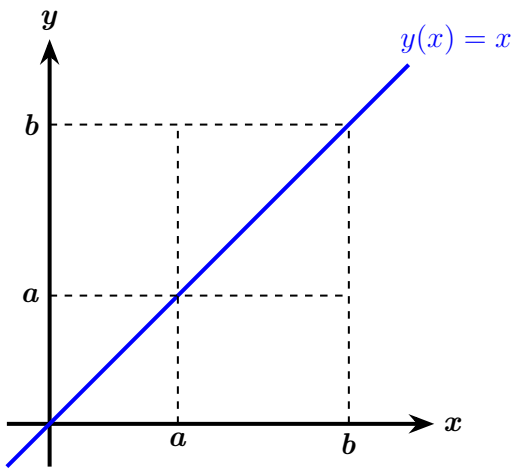
- for the bracketing methods, the root is located within an interval prescribed by a lower and an upper bound
- iterative application of these methods results in closer estimates of the true value of x for $f(x) = 0$
- bracketing methods always converge (assuming the solution is indeed between a and b)
- **open methods** and **fixed point iterations** are based on methods requiring only a single or two values of x
- open methods present a formula to predict the next iterate for solving $f(x) = 0 \ \rightarrow x_{i+1} = g(x_i)$
- these *initializations* do not necessarily bracket the root
- which is kinda great
- caveat: sometimes open methods diverge or move away from the true root
- upside: when open methods converge, they do so quicker than bracketing methods (choose you warrior, when it rains it pours vibes)

fixed point + convergence

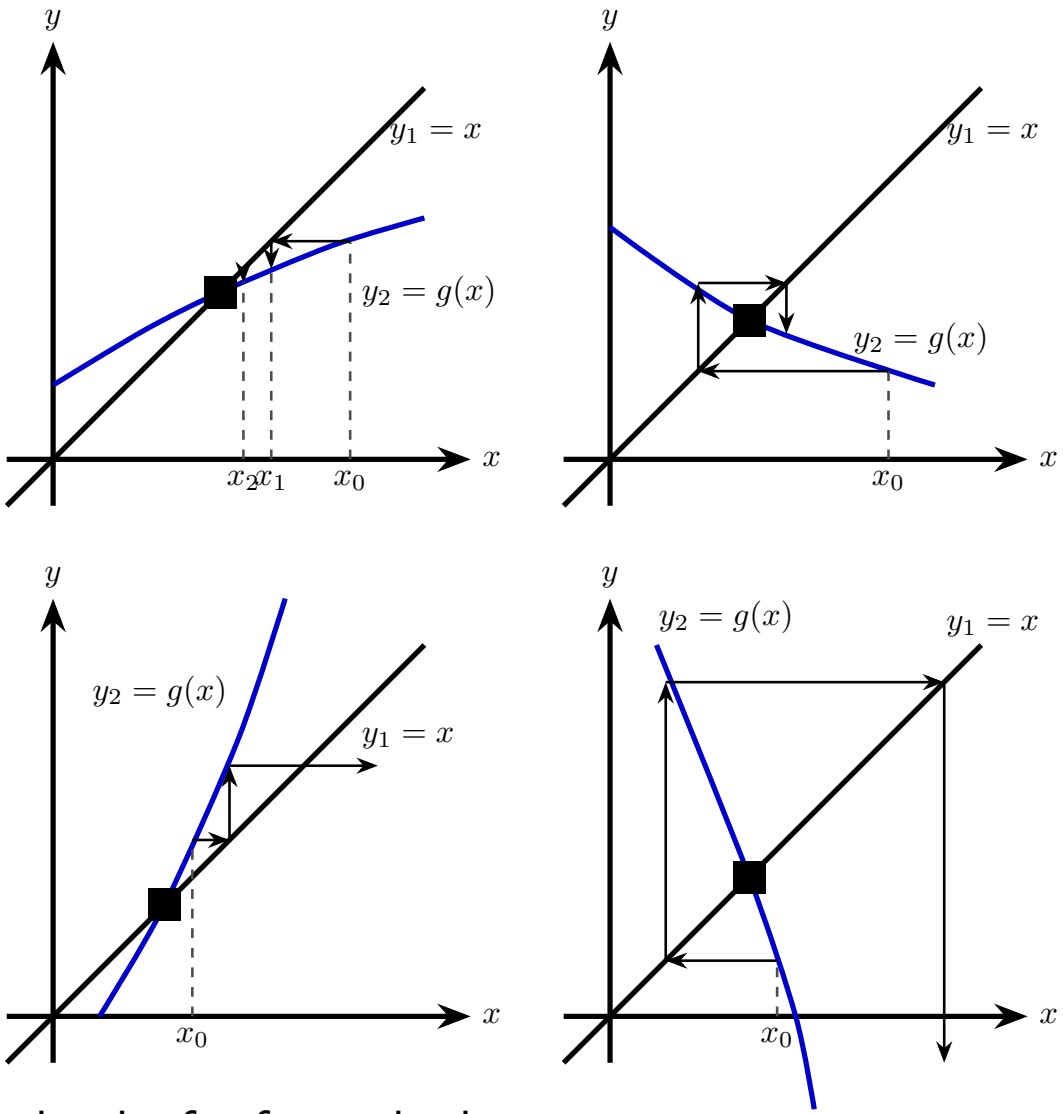
- **definition**: number x is a fixed point for g, if $g(x) = x$
- **example:** find fixed points for $g(x) = x^2 - 2$
- basically a point x is a fixed point of $g(x)$ when the point remains unaltered under the mapping g
- **question**: can we put conditions on $g(x)$ so fp converges? yes

fixed point theorem – existence + uniqueness

1. if $g(x)$ is continuous $\forall x \in [a, b]$ and $x = g(x) \in [a, b]$ then $g(x)$ has at least one fixed point in $[a, b]$ (**existence**)
2. if $g(x)$ is differentiable $\forall x \in [a, b]$ and there exists K s.t. $|g'(x)| \leq K < 1$ then there is a **unique** fixed point x^* in $[a, b]$ and fp converges



graphical interpretation



- iteration cobwebs for fp method
- convergence: figures a) and b), divergence: figures c) and d)

examples and discussions

- does $g(x) = \frac{1}{3}(x^2 - 2)$ have a unique fp for $x \in [-1, 1]$?
- well, the derivative $g(x)$ is $g'(x) = \frac{1}{3}(2x)$ hence

$$|g'(x)| = 2/3|x| \leq 2/3 = K < 1$$

- another example: consider a function $f(x) = x^3 + 4x^2 - 10 = 0$ defined over $[1, 2]$ and analyze convergence
- many ways to write $g(x) = x$ for this function:

$$x = g_1(x) = x - (x^3 + 4x^2 - 10)$$

$$x = g_2(x) = \frac{1}{4}\left(\frac{10}{x} - x^2\right) \qquad \leftarrow x^2 + 4x - \frac{10}{x} = 0$$

$$x = g_3(x) = \left(\frac{10}{x} - 4x\right)^{1/2}$$

$$x = g_4(x) = \frac{1}{2}(-x^3 + 10)^{1/2} \qquad \leftarrow 4x^2 = -x^3 + 10$$

$$x = g_5(x) = \left(\frac{10}{x + 4}\right)^{1/2} \qquad \leftarrow x^2(x + 4) - 10 = 0$$

$$x = g_6(x) = x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}$$

- how to figure out which representation converges? write code that computes $K = \max |g'(x)|$ for all $x \in [1, 2]$ then see if K is less than one for the six functions—only for $g_{5,6}(x)$ the value of $K < 1$

example: fixed point iterations

- rearrange any function $f(x) = 0$ as $x = g(x)$
- **example:** $f(x) = x^2 - 2x + 3 = 0, \quad \rightarrow \quad x = g(x) = 0.5x^2 + 1.5$
- start from any initial point $x_0 = 1$, then $x_{i+1} = g(x_i) = 0.5x_i^2 + 1.5$ until a certain re tolerance is achieved
- another example:

$$f(x) = \sin(x) = 0 \quad \rightarrow \quad f(x) = \sin(x) - x + x = 0 \quad \rightarrow x = \sin(x) + x = g(x)$$

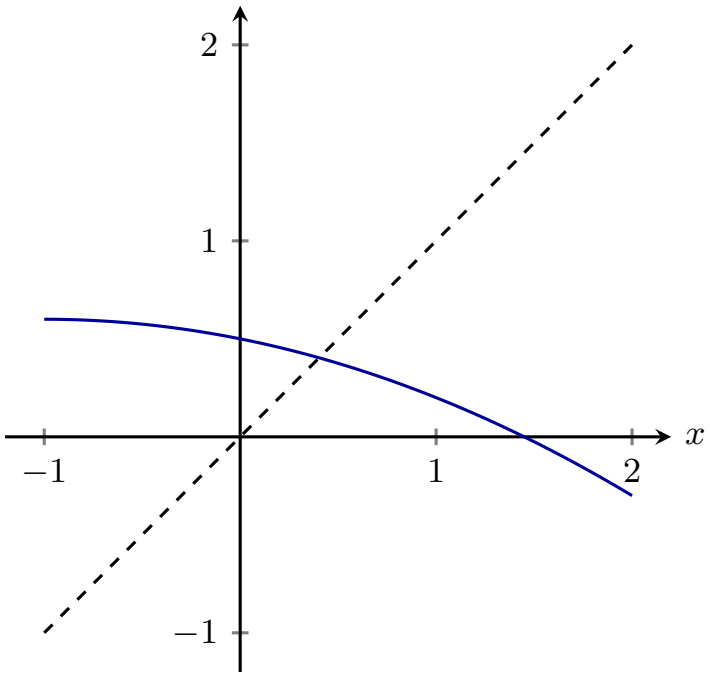
- another one: use fp iteration to solve $f(x) = e^{-x} - x = 0$
- we can write $x_{i+1} = g(x_i) = e^{-x_i}$, with $x_0 = 0$, resulting in:

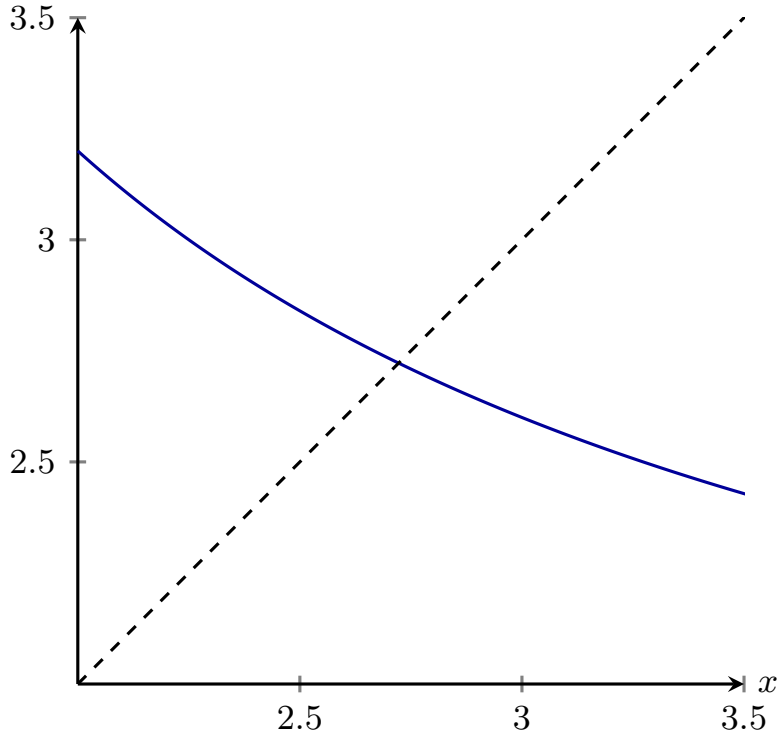
i	x_i	Abs. Error (%)	Rel. Error (%)
0	0	–	100.0
1	1.000000	100.0	76.3
2	0.367879	171.8	35.1
3	0.692201	46.9	22.1
4	0.500473	38.3	11.8
5	0.606244	17.4	6.89
6	0.545396	11.2	3.83
7	0.579612	5.90	2.20
8	0.560115	3.48	1.24
9	0.571143	1.93	0.705
10	0.564879	1.11	0.399

examples

- for each of the following examples, determine an interval $[a, b]$ on which the fp iteration converges

$$x = g_1(x) = \frac{2 - e^x + x^2}{3}, \quad x = g_2(x) = \frac{5}{x^2} + 2$$





- **solution:** need boundaries where absolute derivative is less than 1

newton-raphson method

- also called newton's method but apparently mr. raphson should take credit
- most popular method (plus some modifications) to find solutions/roots of $f(x) = 0$ on earth by far and it's not even close
- **assumption**: assume second derivative exists (function twice diff.)
- **main idea:** assume $\hat{x}^*$ be an approximation of the actual solution x^*
- **method:** try to find a correction term h such that $\hat{x}^* + h$ is a better approximation of x^* than $\hat{x}^*$ and hopefully $\hat{x}^* + h = x^*$
- invoke taylor's theorem:

$$0 = f(x^*) = f(\hat{x}^* + h) = f(\hat{x}^*) + (x - \hat{x}^*) f'(\hat{x}^*) + \frac{(x^* - \hat{x}^*)^2}{2} f''(\xi), \quad \xi \in [x^*, \hat{x}^*]$$

- if $x - \hat{x}^*$ is small, you can ignore the second order term and hence

$$h = x - \hat{x}^* \approx -\frac{f(\hat{x}^*)}{f'(\hat{x}^*)}$$

- which yields the main formula for newton-raphson's method:

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}}$$

analysis of newton-raphson

- assume $f(x) = 0$ has finite J number of solutions/roots x^{j*} for $j = 1, \ldots, J$
- starting from an initialization x_0, this method converges to root x^{j*} which is the closest in euclidean distance to x_0
- so performance is heavily reliant on the initialization point
- **graphical interpretation:** this method can be thought of as computing next iterations at every point for a linear approximation of the function
- how? well recall that the tangent line at $\hat{x}^*$ is

$$f_{lin}(x) = f(\hat{x}^*) + (x - \hat{x}^*)f'(\hat{x}^*)$$

- to find where the function becomes zero, or the x-intercept:

$$0 = f(\hat{x}^*) + (x - \hat{x}^*)f'(\hat{x}^*)$$

hence

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

error analysis

- newton's method can be thought of as a fixed point method as one could write $x_{i+1} = g(x_i)$
- unlike other fp methods, this one cannot progress if $f'(x_i) = 0$
- error analysis of newton's method—define $e_i = x_i - x^*$ (error at iteration i)
- then we can write

$$e_i = x_i - x^* = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} - x^* = \frac{e_{i-1} f'(x_{i-1}) - f(x_{i-1})}{f'(x_{i-1})} = e_i \quad (1)$$

- and recall the following:

$$0 = f(x^*) = f(\hat{x}^* + h) = f(\hat{x}^*) + (x - \hat{x}^*) f'(\hat{x}^*) + \frac{(x^* - \hat{x}^*)^2}{2} f''(\xi), \quad \xi \in [x^*, \hat{x}^*]$$

which can be re-written as

$$0 = f(x^*) = f(x_i - e_i) = f(x_{i-1} - e_{i-1}) = f(x_{i-1}) - e_{i-1} f'(x_{i-1}) + \frac{1}{2} e_{i-1}^2 f''(\xi_{i-1})$$

- combining this equation with (1), yields:

$$\boxed{e_i = \frac{f''(\xi_{i-1}) e_{i-1}^2}{2 f'(x_{i-1})}} \quad \Rightarrow \text{error is prop. to square of previous error or quad. convergence}$$

slow convergence example

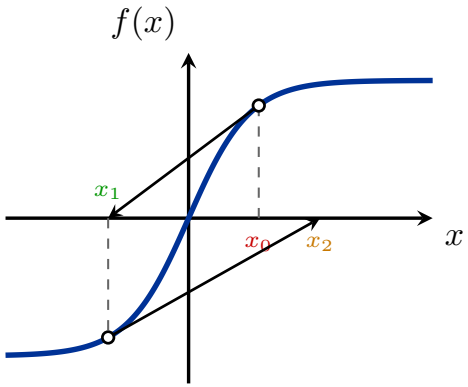
- in some examples, it might take newton's method infinite iterations to converge
- **example:** $f(x) = x^{10} - 1 = 0$ with initial guess $x_0 = 0.5$ results in newton's iteration

$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9}$$

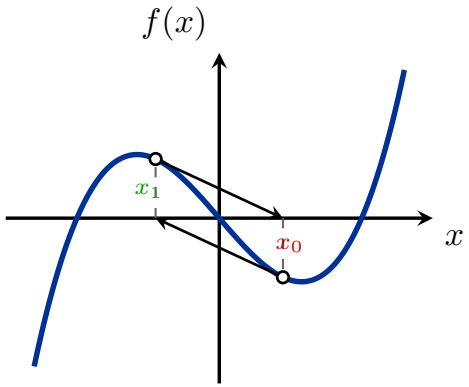
Iteration	***x***
0	0.5
1	51.65
2	46.485
3	41.8365
4	37.65285
5	33.887565
⋮	⋮
∞	1.0000000

- next slide shows examples when newton's method oscillates/fails

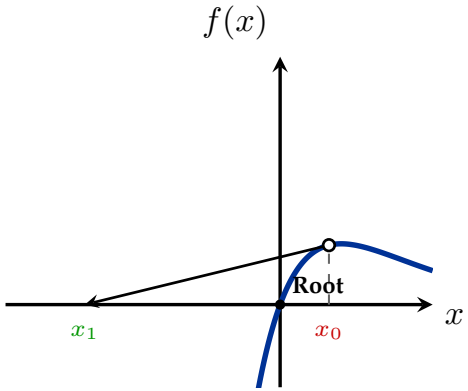
when newton fails



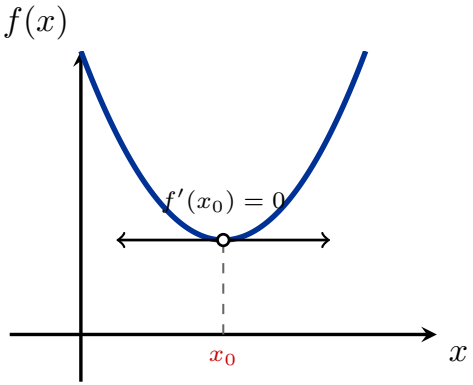
(a) Divergence (Inflection Point)



(b) Infinite Oscillation



(c) Overshoot (Jump Away)



(d) Horizontal Tangent

- figure a: case where an inflection point occurs in root-vicinity
- figure b: demonstrates tendency of the method to oscillate around a local max/min or event sent far away from the root
- figure c: shows an initial guess close to a root can jump far away
- this happens because newton's method wants to avoid a zero-slope at all costs
- no general convergence criterion for newton's method
- use expert knowledge

one more example

- find the square root of any positive number m via newton's method
- **solution:** the root finding problem can be written as $f(x) = x^2 - m = 0$
- then, newton's iteration can be written as

$$x_{i+1} = x_i - \frac{x_i^2 - m}{2x_i} = \frac{1}{2}\left(x_{i-1} + \frac{m}{x_{i-1}}\right)$$

- assuming that $m = 16$ and $x_0 = 1$, we get:

Iteration i	**Value** x
1	8.50000000000000
2	5.1911764705882
3	4.1366647225462
4	4.0022575247985
5	4.0000006366929
6	4.0000000000000
7	4.0000000000000
8	4.0000000000000

the world is a function of more than variable

- what kind of function is only a function of one variable??
- what if we're solving
  $f_1(x_1, x_2, x_3, \ldots, x_n) = 0, f_2(x_1, x_2, \ldots) = 0, \ldots, f_n(x_1, x_2, \ldots) = 0$?
- we can write this system of equations as $\boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^n$
- we follow a similar procedure as in the single-variable case
- first, define the jacobian matrix $\boldsymbol{D}_{\boldsymbol{f}(\boldsymbol{x}_0)} \in \mathbb{R}^{n \times n}$ as the matrix containing the partials of f evaluated at $\boldsymbol{x}_0$
- obtain the linearized taylor series approximation:

$$\boldsymbol{f}(\boldsymbol{x}) \approx 0 = \boldsymbol{f}(\boldsymbol{x}_0) + \boldsymbol{D}_{\boldsymbol{f}(\boldsymbol{x}_0)}(\boldsymbol{x} - \boldsymbol{x}_0)$$

or

$$\boldsymbol{x} = \boldsymbol{x}_0 - \boldsymbol{D}^{-1}_{\boldsymbol{f}(\boldsymbol{x}_0)} \boldsymbol{f}(\boldsymbol{x}_0)$$

or more generally

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{D}^{-1}_{\boldsymbol{f}(\boldsymbol{x}_i)} \boldsymbol{f}(\boldsymbol{x}_i)$$

example

- find the solutions to

$$f_1(x_1, x_2) = x_1^2 + x_2^2 - 4 = 0 \quad , \quad f_2(x_1, x_2) = 4x_1^2 - x_2^2 - 4 = 0$$

  with an initialization $\boldsymbol{x}_0 = [1, 1]^\top$
- jacobian can be computed as

$$\boldsymbol{D}_{\boldsymbol{f}(\boldsymbol{x})} = \begin{bmatrix} 2x_1 & 2x_2 \\ 8x_1 & -2x_2 \end{bmatrix}$$

- the inverse is

$$\boldsymbol{D}^{-1}_{\boldsymbol{f}(\boldsymbol{x})} = \frac{1}{-20x_1x_2} \begin{bmatrix} -2x_2 & -2x_2 \\ -8x_1 & 2x_1 \end{bmatrix}$$

- then

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{D}^{-1}_{\boldsymbol{f}(\boldsymbol{x}_i)} \boldsymbol{f}(\boldsymbol{x}_i)$$

$$\boldsymbol{x}_{i+1} = \begin{bmatrix} x_{1,i} - \dfrac{5x_{1,i}^2 - 8}{10x_{1,i}} \\ x_{2,i} - \dfrac{5x_{2,i}^2 - 12}{10x_{2,i}} \end{bmatrix}$$

- for $i = 0$, we get

$$\boldsymbol{x}_1 = \begin{bmatrix} 1.3 \\ 1.7 \end{bmatrix} \rightarrow \boldsymbol{x}_2 = \begin{bmatrix} 1.26 \\ 1.55 \end{bmatrix} \rightarrow \boldsymbol{x}_3 = \cdots$$

secant method

- recall the newton's method iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- unfortunately this requires knowing the value of derivative of $f(x)$
- computing derivatives can often be more difficult than evaluating the function itself
- secant method overcomes this issue by approximating the derivative as

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

yielding:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}, \quad i \geq 1$$

- this method requires two initial points: x_0 and x_1 which is no big deal

modified secant method

- some people hate the approximation

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

- need to find another way to approximate the derivative that does not require two initial points
- **solution:**

$$f'(x_i) \approx \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

where δx_i is like a little perturbation to x_i yielding the modified secant method:

$$x_{i+1} = x_i - f(x_i)\frac{\delta x_i}{f(x_i + \delta x_i) - f(x_i)}$$

- no free lunch though: choice of δx_i is not trivial
- if (δx_i) is too big, method diverges; if too small, method gets lost in round-off error
- method is still nice if you don't wanna use two initializations (secant) or you don't wanna use the derivative (newton)

secant + false position method

- recall the iteration for the false position method:

$$x^* = b - \frac{f(b)(a-b)}{f(a) - f(b)}$$

compared to

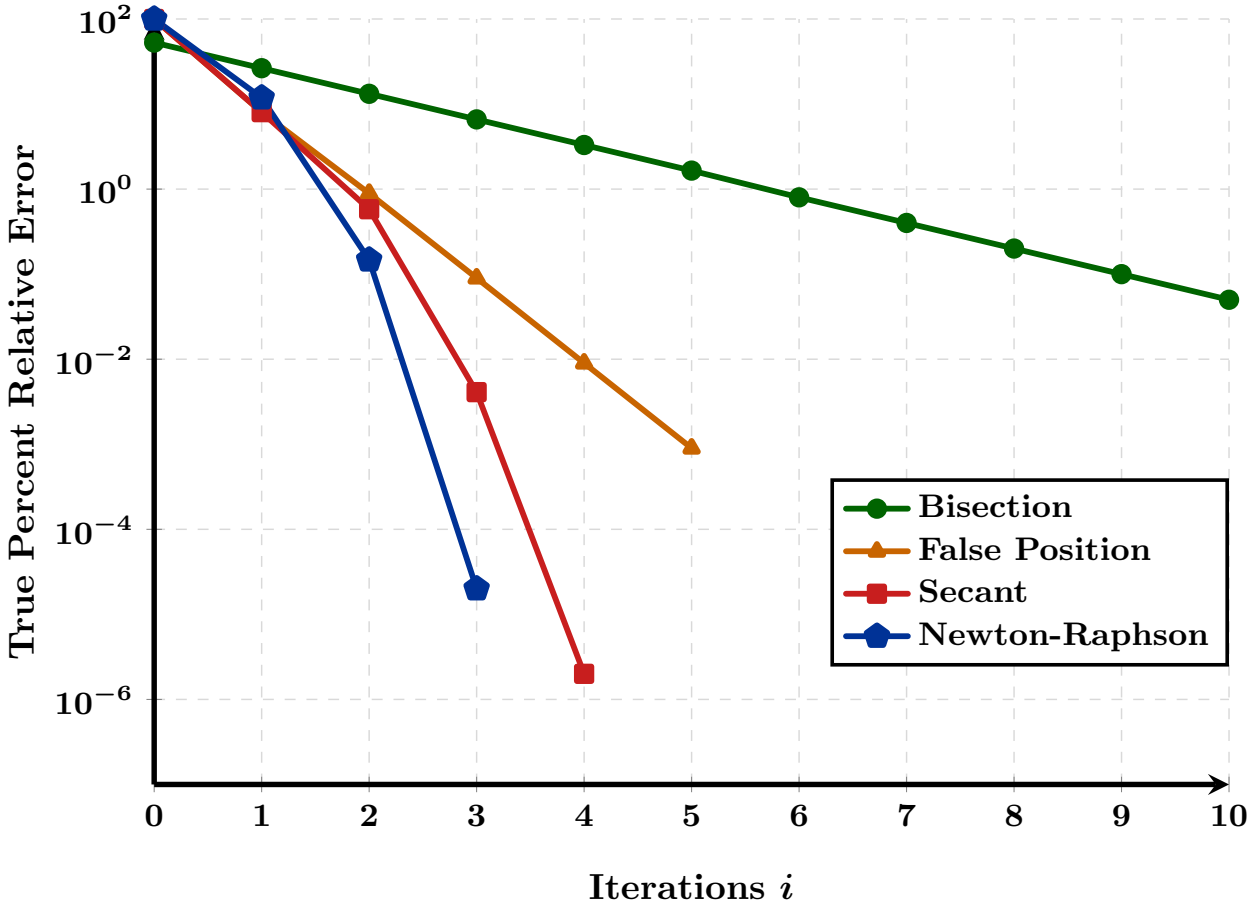
$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}, \quad i \geq 1$$

for the secant method

- not the similarity—they're basically identical
- both use two initial estimates to compute an approximation of the slope of the function
- a critical difference between the two: how one of the initial values is replaced by the new estimate
- false position always converges as the root is kept within the bracket
- secant method can yield two initial values that lie on the same side of the root leading to divergence

comparison of all methods

- comparison of the true percent relative errors for the methods to determine the solution of $f(x) = e^{-x} - x = 0$:
- actual solution is $x^* \approx 0.567143$
- initial conditions are:
  - newton-raphson and secant: initial guess $x_0 = 0$
  - bisection and false position: bracketing interval $[0, 1]$



brent's method

- convergent methods: bracketing methods
- fast but possibly divergent: secant + newton-raphson
- would be nice to combine them all amirite?
- this is what brent's method basically does

An algorithm with guaranteed convergence for finding a zero of a function

R. P. Brent*

Computer Science Department, Stanford University, Stanford, California 94305, USA

An algorithm is presented for finding a zero of a function which changes sign in a given interval. The algorithm combines linear interpolation and inverse quadratic interpolation with bisection. Convergence is usually superlinear, and is never much slower than for bisection. ALGOL 60 procedures are given.

(Received August 1970, Revised March 1971)

1. Introduction

Let f be a real-valued function, defined on the interval $[a, b]$, with $f(a)\,f(b) \leqslant 0$. f need not be continuous on $[a, b]$: for example, f might be a limited-precision approximation to some continuous function (see Forsythe, 1969). We want to find an approximation $\hat{\zeta}$ to a zero ζ of f, to within a given positive tolerance 2δ, by evaluating f at a small number of points. Of course, if f is discontinuous then there may be no zero in $[a, b]$, so we shall be satisfied if f takes both non-negative and non-positive values in $[\hat{\zeta} - 2\delta, \hat{\zeta} + 2\delta] \cap [a, b]$. Clearly, such a $\hat{\zeta}$ may always be found by bisection in about $\log_2[(b - a)/\delta]$ steps, and this is the best that we can do for arbitrary f. We shall describe an algorithm which is never much slower than bisection, but which has the advantage of superlinear convergence to a simple zero ζ, if we ignore rounding errors and suppose that f is continuously differentiable near ζ. This means that, in practice, convergence is often much faster than for bisection.

mation so far to ζ, a is the previous value of b, and ζ must lie between b and c (initially $a = c$).

If $f(b) = 0$ then we are finished (the ALGOL procedure given by Dekker (1969) does not recognise this case, and can take a large number of small steps if f vanishes on an interval, which may happen because of underflow).

If $f(b) \neq 0$ then let $m = \frac{1}{2}(c - b)$. We prefer not to return with $\hat{\zeta} = \frac{1}{2}(b + c)$ as soon as $|m| \leqslant 2\delta$, for if superlinear convergence has set in then b is probably a much better approximation to ζ than $\frac{1}{2}(b + c)$ is. Instead, we return with $\hat{\zeta} = b$ if $|m| \leqslant \delta$ (so the error is no more than δ if, as is often true, f is nearly linear between b and c), and otherwise interpolate (or extrapolate) f linearly between a and b, giving a new point i (see Section 4 for inverse quadratic interpolation). To avoid the possibility of overflow or division by zero we find i as $b + p/q$, and the division is not performed if $2|p| \geqslant 3|m.q|$, for then i is not needed anyway. The reason why the simpler criterion $|p| \geqslant |m.q|$ is not used is explained in Section 4.

the actual algorithm is detailed

Appendix: Algol 60 procedures

```
real procedure zero (a, b, macheps, t, f);
value a, b, macheps, t; real a, b, macheps, t;
real procedure f;
  begin comment:
```

Procedure *zero* returns a zero x of the function f in the given interval $[a, b]$, to within a tolerance $6macheps\ |x| + 2t$, where *macheps* is the relative machine precision and t is a positive tolerance. The procedure assumes that $f(a)$ and $f(b)$ have different signs;

```
  real c, d, e, fa, fb, fc, tol, m, p, q, r, s;
  fa := f(a); fb := f(b);
  int: c := a; fc := fa; d := e := b − a;
  ext: if abs(fc) < abs(fb) then
    begin a := b; b := c; c := a;
    fa := fb; fb := fc; fc := fa
    end;
  tol := 2 × macheps × abs(b) + t; m := 0·5 × (c − b);
  if abs(m) > tol ∧ fb ≠ 0 then
```

424

```
int: c := a; fc := fa; ec := ea; d := e := b − a;
ext: if (ec ⩽ eb ∧ pwr2(abs(fc), ec − eb) < abs(fb))
      ∨ (ec > eb ∧ pwr2(abs(fb), eb − ec) ⩾ abs(fc)) then
  begin a := b; fa := fb; ea := eb;
        b := c; fb := fc; eb := ec;
        c := a; fc := fa; ec := ea
  end;
tol := 2 × macheps × abs(b) + t; m := 0·5 × (c − b);
if abs(m) > tol ∧ fb ≠ 0 then
  begin if abs(e) < tol ∨
  (ea ⩽ eb ∧ pwr2(abs(fa), ea − eb) ⩽ abs(fb)) ∨
  (ea > eb ∧ pwr2(abs(fb), eb − ea) ⩾ abs(fa)) then
  d := e := m else
    begin s := pwr2(fb, eb − ea)/fa; if a = c then
      begin p := 2 × m × s; q := 1 − s end
    else
      begin q := pwr2(fa, ea − ec)/fc;
      r := pwr2(fb, eb − ec)/fc;
      p := s × (2 × m × q × (q − r) − (b − a) × (r − 1));
      q := (q − 1) × (r − 1) × (s − 1)
      end;
```

The Computer Journal

- can be found here
  https://maths-people.anu.edu.au/~brent/pd/rpb005.pdf

solving polynomials of degree n

- consider this polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0$$

fundamental theorem of algebra

- there is exactly n roots for an nth degree polynomial, including multiplicity meaning that

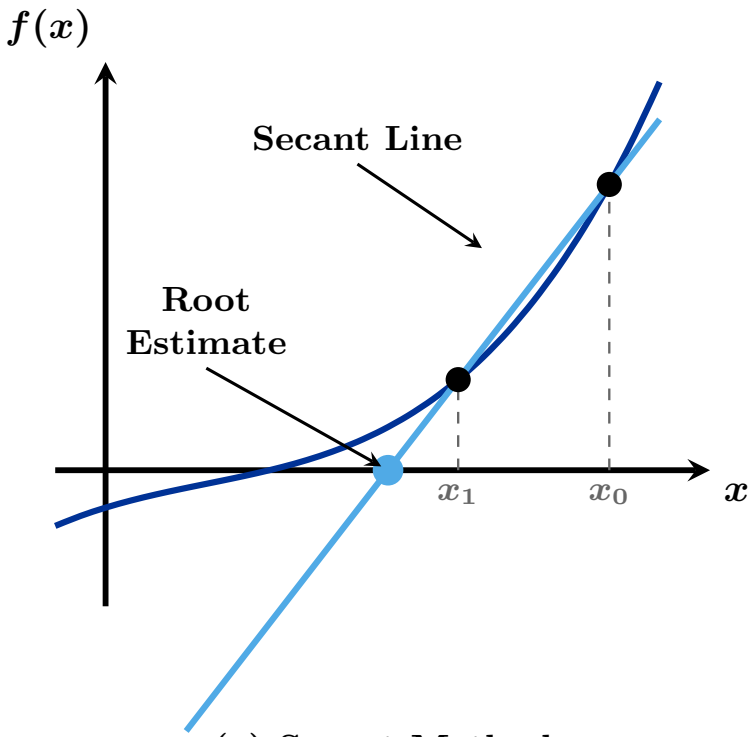
$$p(x) = a_n (x - x_1)^{m_1} (x - x_2)^{m_2} \cdots (x - x_k)^{m_k} = 0, \quad \sum_{i=1}^{k} m_i = n$$

  m_k: multiplicity of the kth root (how many times its repeated)
- if the coefficients a_i are complex $\Rightarrow$ complex roots exist
- if the coefficients a_i are all real, then some roots can be complex
- if the coefficients a_i are all real and n is odd, then at least one root is real
- complex roots appear in conjugate terms (e.g., for a third degree polynomial, you gotta have at least one real root)

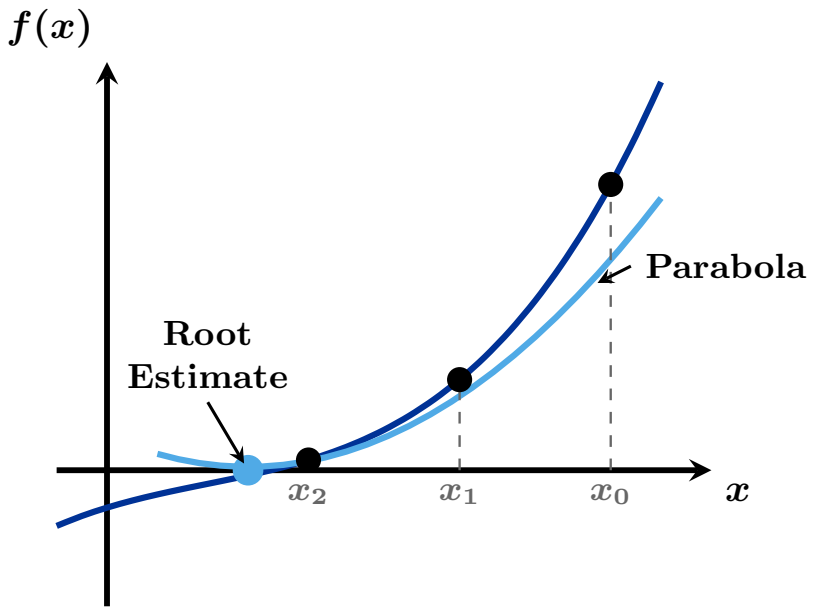
applying methods to poly roots

- we can apply all the methods we studied (bisection, false position, newton, secant) to find roots for polynomials
- reliability and performance of these methods depend on what kind of roots we have
- only real roots: all the methods could work
- finding good initial guesses when you have complex roots (and coefficients) complicates bracketing/open methods
- complex roots prevent the usage of vanilla bracketing methods
- **why?** defining a bracket and sign change does not translate to complex numbers
- newton-raphson can be used to find complex roots but susceptible to convergence issues
- to that end, two methods are widely used to deal with this: the müller and bairstow methods
- müller method: like the secant but instead of a line, obtain a parabola
- bairstow's method: somewhere between müller and newton-raphson
- the methods are detailed but we have bigger fish to fry

muller's method vs secant's method



(a) Secant Method



(b) Müller's Method

- secant method approximates the function using a straight line
- then it estimates the root where this line crosses the x-axis
- müller's method approximates the function using a parabola passing through
- initialization + convergence: secant method requires 2 initial guesses; müller's requires 3

matlab commands

- matlab has a bunch of commands to compute roots of equations
- most of these apply to functions of one variables
- to use matlab to solve nonlinear systems of equations $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, you need to use `fsolve`
- command `fsolve` requires an initial guess so solution will be the root closest to the guess

calling solvers

```
>> x0=[0 1.3];
>> x=fzero(@(x) x^10-1,x0)
x = 1
>> x0=[-1.3 0];
>> x=fzero(@(x) x^10-1,x0)
x =
-1
>> x0=0;
>> x=fzero(@(x) x^10-1,x0)
x =
-1
```

```
>> x0=0;
>> option=optimset('DISP','ITER');
>> x=fzero(@(x) x^10-1,x0,option)
Func-count x                          f(x)       Procedure
     1              0                  -1       initial
     2     -0.0282843                  -1       search
     3      0.0282843                  -1       search
     4          -0.04                  -1       search
     •
     •
     •
    21           0.64           -0.988471       search
    22      -0.905097           -0.631065       search
```

- what method does `fzero` implement? from matlab:

The fzero command is a function file. The algorithm, created by T. Dekker, uses a combination of bisection, secant, and inverse quadratic interpolation methods. An Algol 60 version, with some improvements, is given in Brent, R., ***Algorithms for Minimization Without Derivatives****, Prentice-Hall, 1973.*

module summary

- what did we learn?
- a lot of methods to solve $f(x) = 0$ for a single variable
- but also methods to solve $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ for n variables
- some methods always converge, some diverge under certain conditions
- no way to know unless you actually try things out on your applications
- the more you ... around, you more you find out

method	**type**	**guesses**	**convergence**	**stability**	**prog.**	**comments**
direct	analytical	—	—	—	—	—
graphical	visual	—	—	—	—	imprecise
bisection	bracketing	2	slow	always	easy	—
false-position	bracketing	2	slow/medium	always	easy	—
fixed-point iter.	open	1	slow	possibly divergent	easy	—
newton-raphson	open	1	fast	possibly divergent	easy	requires evaluation of $f'(x)$
secant	open	2	medium/fast	possibly divergent	easy	initial guesses do not have to bracket the root
brent	hybrid	1 or 2	medium	always	moderate	robust
müller	polynomials	2	medium/fast	possibly divergent	moderate	—
bairstow	polynomials	2	fast	possibly divergent	moderate	—