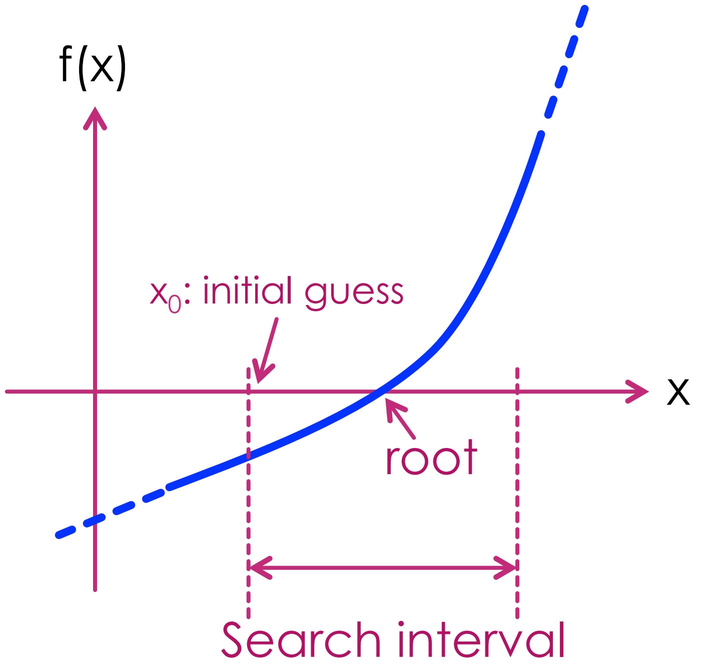
Solving Equations (1D)

Solving equations: f(x)=0

✓ **Real Roots of real-valued functions over the interval (a, b)**

- One-dimensional functions are concerned (one variable)
- Assume that the function is continuous and endpoints are excluded
- Complex roots are not easy
- For special types of functions, complex roots are easily found
    - Polynomials → **Laguerre's Method** for complex roots
- Real roots have various algorithms (criteria: speed of search)
    - General Lemma of **root bracketing**
    - Bisection method, Trisection method
    - Secant method
    - Brent's method
    - Newton-Raphson's method
    - Generalized Taylor method



Solving equations: f(x)=0

Root-finding algorithm

https://en.wikipedia.org/wiki/Root-finding_algorithm

From Wikipedia, the free encyclopedia

In mathematics and computing, a root-finding algorithm is an algorithm for finding roots of continuous functions. A root of a function f, from the real numbers to real numbers or from the complex numbers to the complex numbers, is a number x such that f(x) = 0. As, generally, the roots of a function cannot be computed exactly, nor expressed in closed form, root-finding algorithms provide approximations to roots, expressed either as floating point numbers or as small isolating intervals, or disks for complex roots (an interval or disk output being equivalent to an approximate output together with an error bound).

Solving an equation f(x) = g(x) is the same as finding the roots of the function h(x) = f(x) – g(x). Thus root-finding algorithms allow solving any equation defined by continuous functions. However, most root-finding algorithms do not guarantee that they will find **all the roots**; in particular, if such an algorithm does not find any root, that does not mean that no root exists.

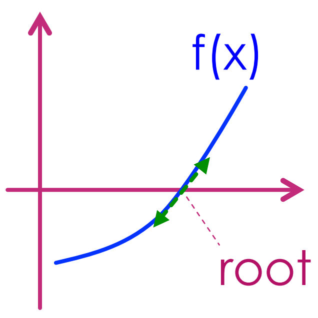
Most numerical root-finding methods use **iteration**, producing a sequence of numbers that hopefully converge towards the root as a limit. They require one or more **initial guesses** of the root as starting values, then each iteration of the algorithm produces a successively more accurate approximation to the root. Since the iteration must be stopped at some point these methods produce an approximation to the root, not an exact solution. Many methods compute subsequent values by evaluating an auxiliary function on the preceding values. The limit is thus a fixed point of the auxiliary function, which is chosen for having the roots of the original equation as fixed points, and for converging rapidly to these fixed points.

Solving equations: f(x)=0

✓ **Visual understanding of roots of f(x)**

- Note how the sign of f(x), f'(x), f''(x), … behaves in a neighborhood about the root

In a neighborhood of the root, we can have various behaviors for the function



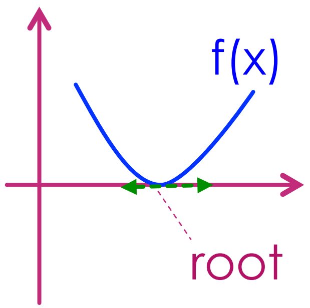
$f(r) = 0$

$f'(r) > 0$

$f''(r) > 0$

Simple root

$f(x) \sim (x-r)$



$f(r) = 0$

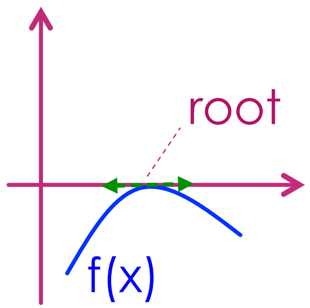
$f'(r) = 0$

$f''(r) > 0$

Even root

$f(x) \sim (x-r)^2$

tangent



$f(r) = 0$

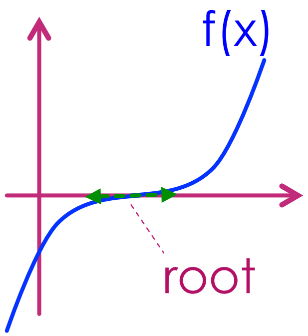
$f'(r) = 0$

$f''(r) < 0$

Even root

$f(x) \sim -(x-r)^4$

tangent



$f(r) = 0$

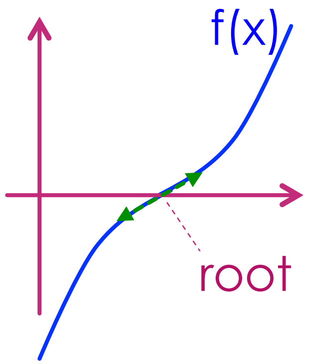
$f'(r) = 0$

$f''(r) = 0$

Odd root

$f(x) \sim (x-r)^3$

Stationary point of inflection



$f(r) = 0$

$f'(r) > 0$

$f''(r) = 0$

Simple root

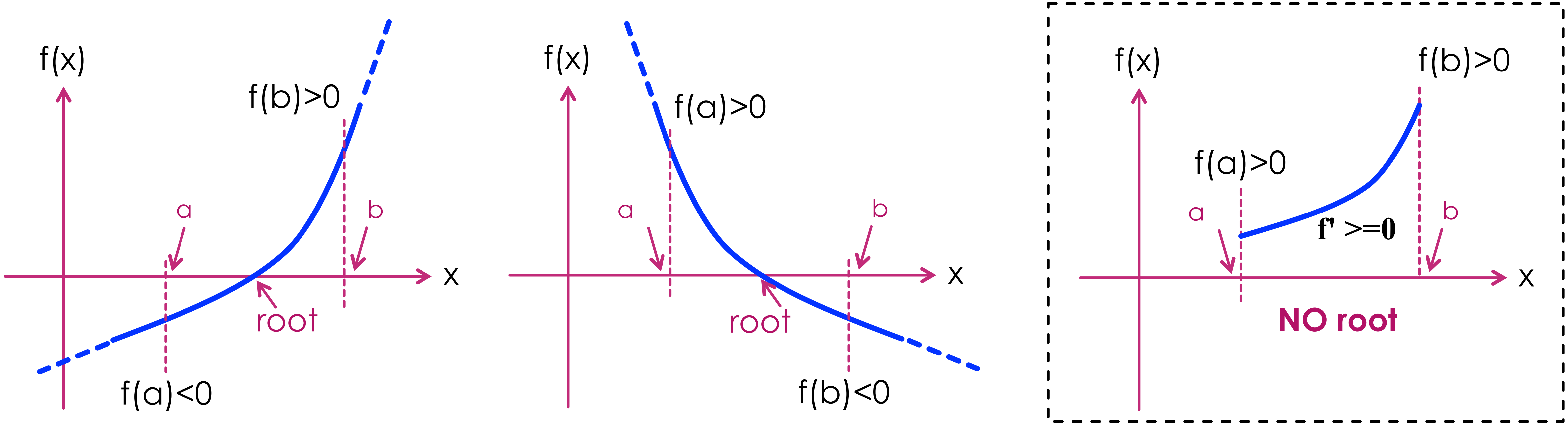
$f(x) \sim (x-r)$

Non-stationary point of inflection

General Lemma: f(x)=0

✓ **Consider a <u>continuous</u> function f(x) over the interval (a, b)**

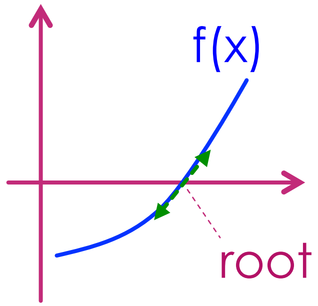
- If f(a).f(b)<0 then f(x) **MUST** have <u>at least one root</u> in this interval
- **Exact** number of roots is **not** guaranteed
  - Only the existence of **one** root is guaranteed
  - The interval can become <u>arbitrarily small</u>, hence an accurate estimate of root
- if f(a) > 0 and f'(x) >= 0 for **all x**, then f(x) has **NO roots in (a, b)**



Bi-section method

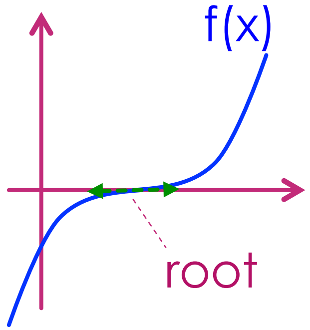
✓ **Bracketing and Binary search**

- Fast and effective for simple or odd roots
- Order ~ **log**
- Binary search
- Simplest method based on general lemma
- Does not work for roots with even multiplicity $(x-r)^{2k}$



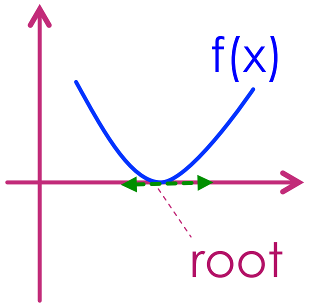
Simple root

$f(x) \sim (x-r)$



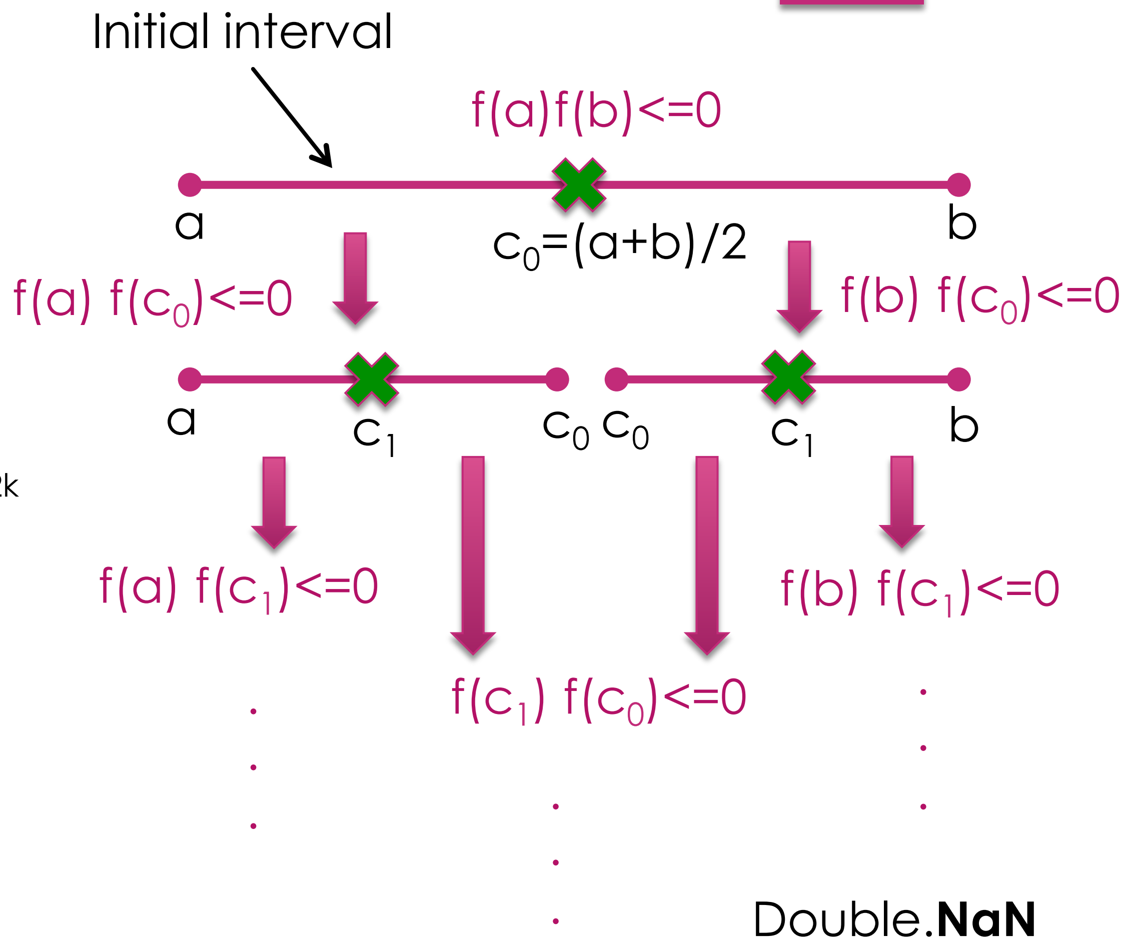
Odd root

$f(x) \sim (x-r)^3$



Even root

$f(x) \sim (x-r)^2$



Double.**NaN**

If there is no roots, return NaN

Tip: use recursion to implement binary search of Bi-section method

Bi-section method

✓ **Estimation of error**

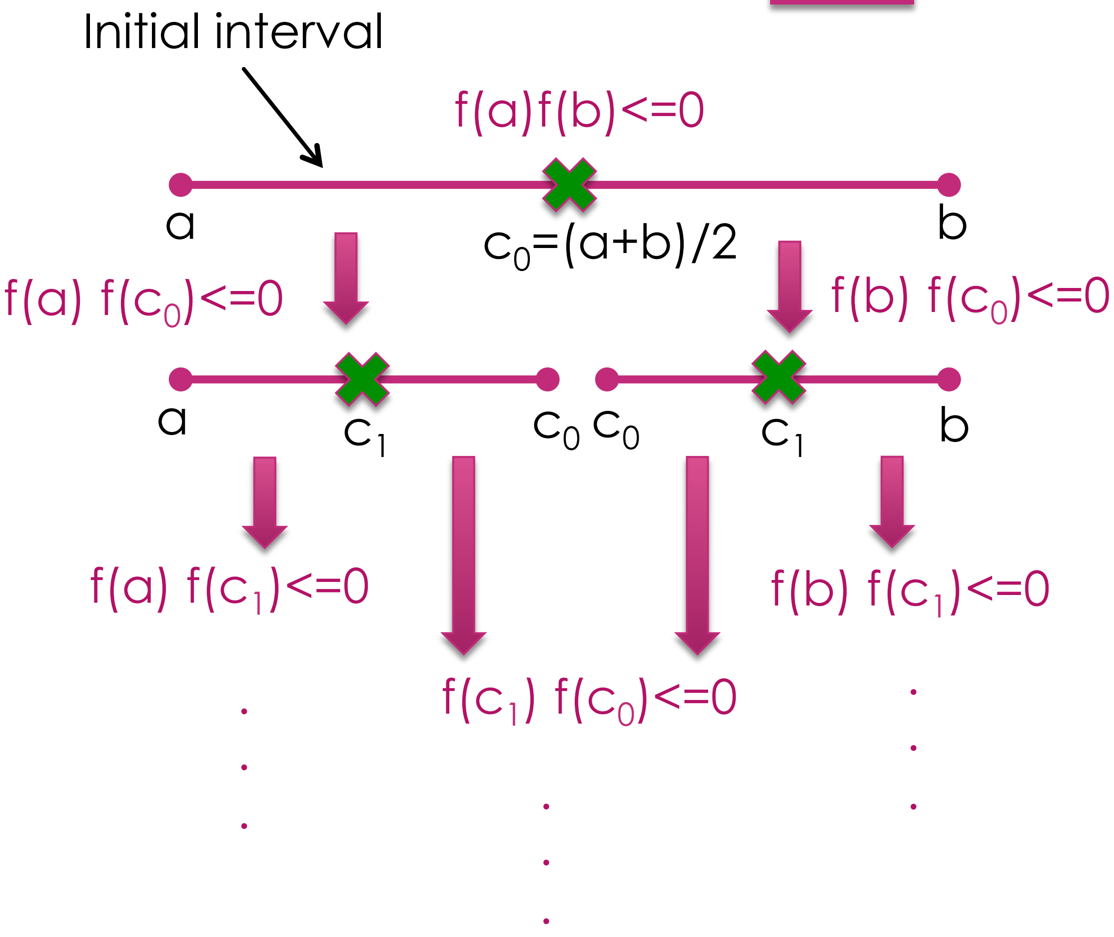
- Absolute Error is reduced by half after each step
- How many steps do we need?
  - Depends on the error
- We can define a relative error (sequences)

$$\mathrm{RelError} = \varepsilon_n = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right|$$

- The number of steps is then given by:

$$\varepsilon_n = \frac{b-a}{2^n} \quad \Rightarrow \quad \boxed{n = \log_2\left(\frac{b-a}{\varepsilon}\right)}$$

- What if the root is much closer to one end of the interval? We can set **an initial guess** for the root.



Map (a, b) interval to (0, 1) so that b-a = 1

Bi-section method

✓ **Java implementation**

- Binary search can be implemented **recursively**
- **Step 1:**
  - Define an interface for real function
  - Use it as a functional interface
- **Step 2:**
  - Define a class for implementing bi-section
  - Look for only one root
  - Parameters: Initial interval + initial guess for root
- **Step 3:**
  - Implement the recursive search
  - Setup the stop criteria

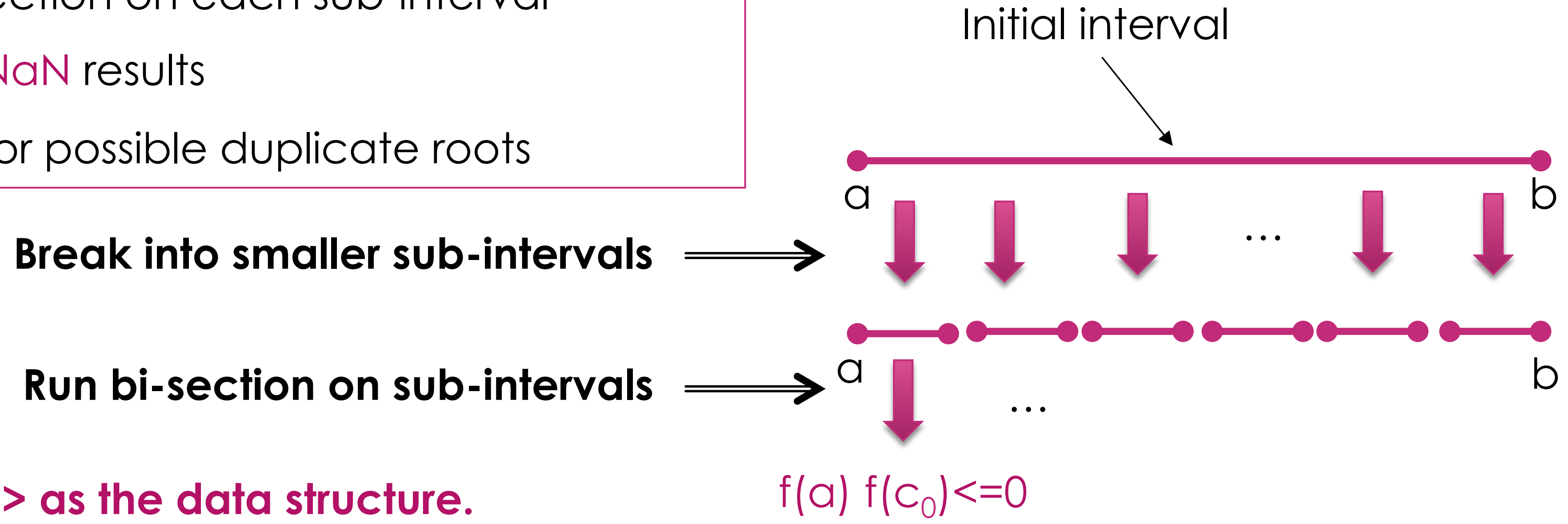
Bi-section method

✓ **How to find <u>All roots</u> in an interval?**

- Bi-section is very fast
  - Requires minimum calculations compared to other methods
- Except for the **tangent roots**

- **Step 1**: break the interval into smaller sub-intervals
- **Step 2**: run bi-section on each sub-interval
- **Step 3**: ignore NaN results
- **Step 4**: check for possible duplicate roots

Collection Framework
- ArrayList<Doubles>
- Set<Doubles>



Tip: use <u>List</u><Double> as the data structure.

Tri-section method

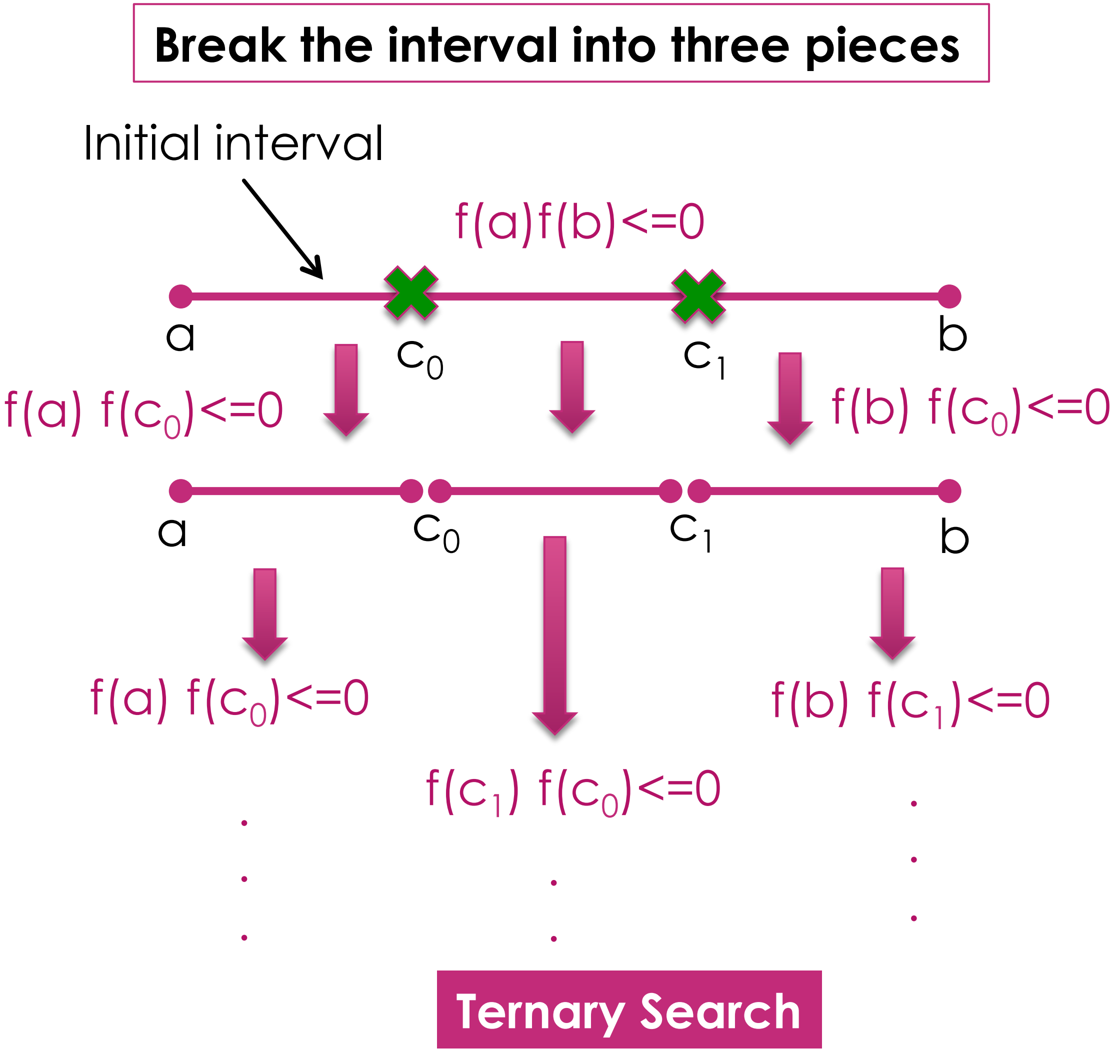
✓ **Ternary search**

- Results in faster convergence
- Implementation is similar to bi-section
  - **Recursion**
- Absolute Error is reduced by half after each step
- How many steps do we need?
  - Depends on the error
- We can define a relative error (sequences)

$$\text{RelError} = \varepsilon_n = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right|$$

- The number of steps is then given by:

$$\varepsilon_n = \frac{b-a}{3^n} \Rightarrow \boxed{n = \log_3\left(\frac{b-a}{\varepsilon}\right)}$$



Secant method

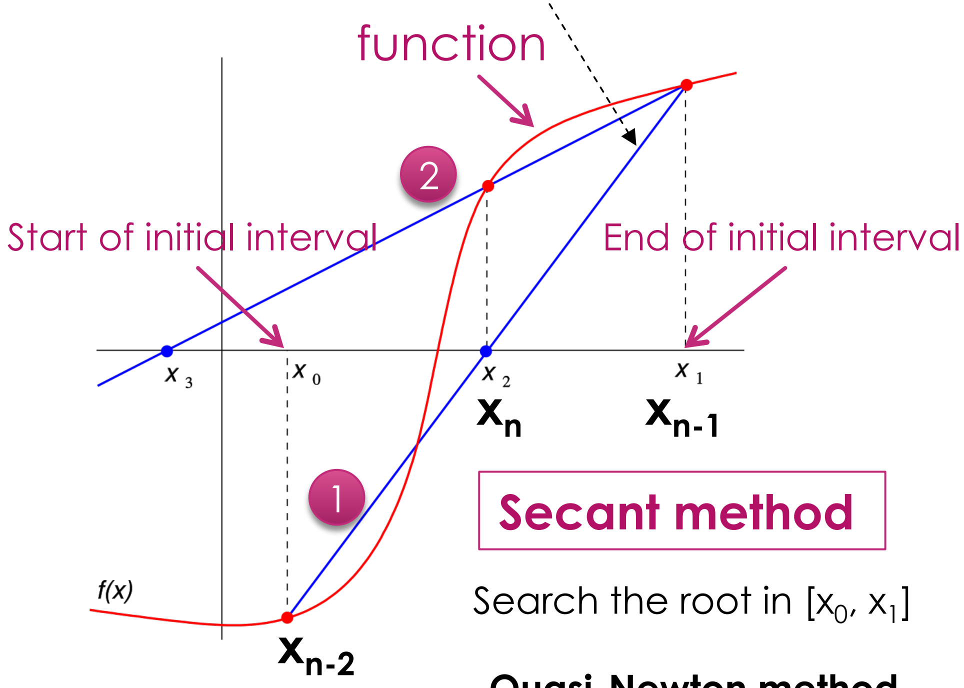
$$y - f(x_{n-1}) = \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}(x - x_{n-1})$$

Brent's method

From Wikipedia

From Wikipedia, the free encyclopedia

In numerical analysis, Brent's method is a root-finding algorithm **combining** the bisection method, the **secant method** and inverse quadratic interpolation. It has the reliability of bisection but it can be as quick as some of the less-reliable methods. The algorithm tries to use the potentially fast-converging secant method or inverse quadratic interpolation <u>if possible</u>, but <u>it falls back to the more robust bisection method if necessary</u>. Brent's method is due to **Richard Brent** and builds on an earlier algorithm by Theodorus Dekker. Consequently, the method is also known as the **Brent–Dekker method**.



Secant method

Search the root in $[x_0, x_1]$

Quasi-Newton method

Next guess

Previous guesses

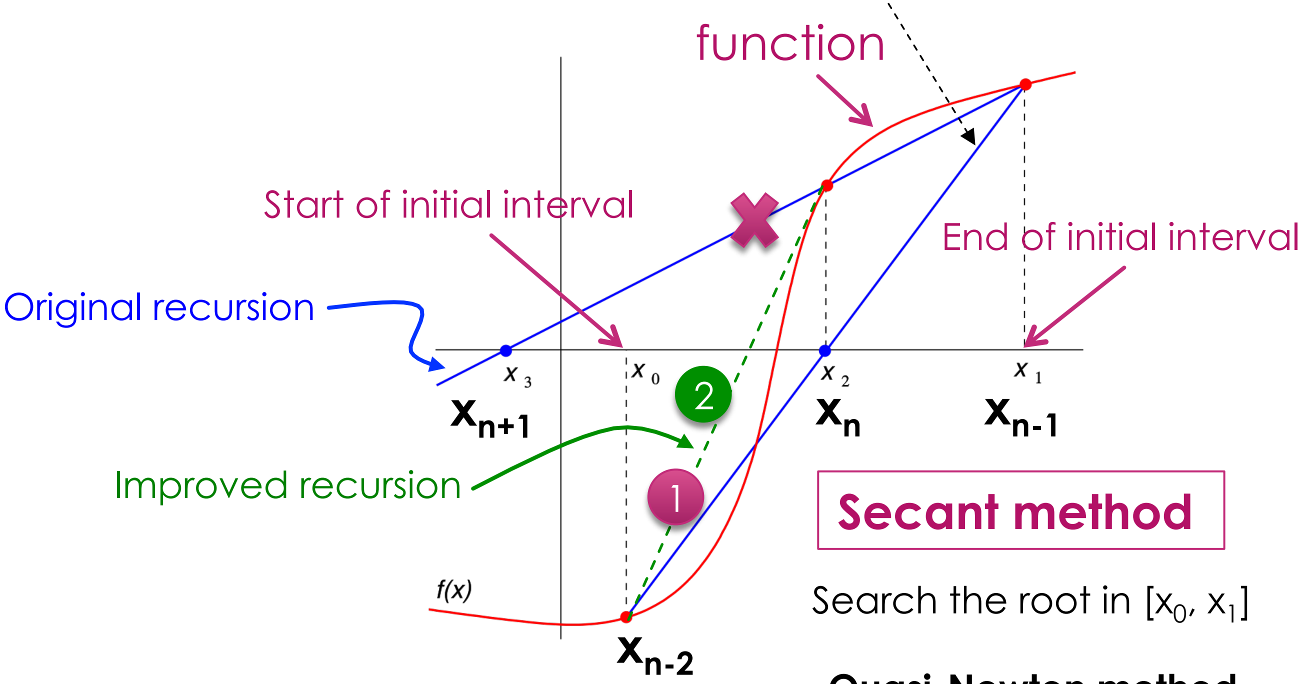
Recursion formula: $x_n = x_{n-1} - f(x_{n-1})\dfrac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$

Tip: use <u>recursion</u> to implement <u>Secant</u> method

Improved Secant method

- **Improved recursion**
  - If the start bracketing interval is $[x_0, x_1]$, we want to make sure the recursion remains in this interval.
  - x_2 is guaranteed to be in $[x_0, x_1]$ interval, but the next iterations are not.
  - Each step of the secant method applies to a valid bracket
  - Secant is guaranteed to converge to the root in this interval
  - This method is also known as "**False Position Method**"

$$y - f(x_{n-1}) = \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}(x - x_{n-1})$$



Secant method

Search the root in $[x_0, x_1]$

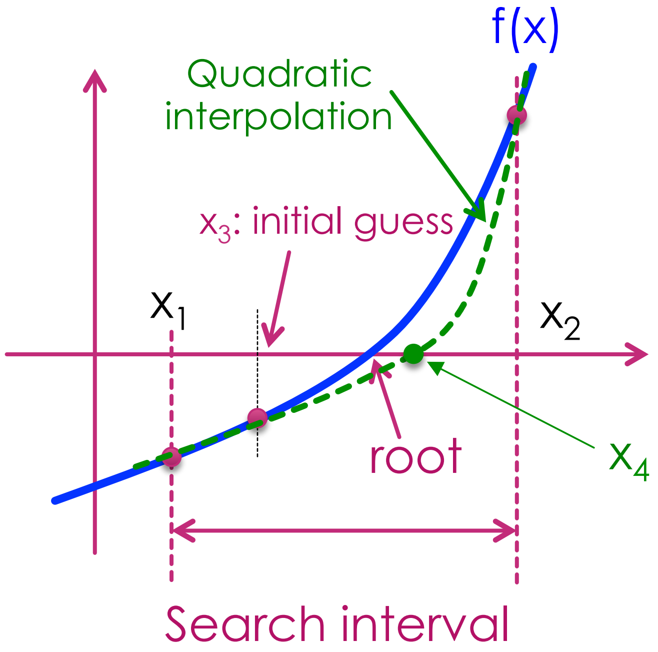
Quasi-Newton method

Tip: use recursion to implement Secant method

Brent's method

✓ **Based on Lagrange interpolating polynomial**

- Given three points x_1, x_2, x_3
    - Fit x as a quadratic polynomial of f(x)
- Root must have been already **bracketed**
- An initial guess can be used



Secant method → Linear interpolation

Brent's method → Quadratic

$y_1 = f(x_1)$, $y_2 = f(x_2)$, $y_3 = f(x_3)$

Quadratic interpolation for x

$$x = p(y) = \prod_{i,j \neq k} \frac{(y - y_i)(y - y_j)}{(y_k - y_i)(y_k - y_i)} x_k$$

i,j,k=1,2,3

Three terms

Find zero crossing point y = 0

$$x_4 = \prod_{i,j \neq k} \frac{y_i \, y_j}{(y_k - y_i)(y_k - y_i)} x_k$$

Recursion with (x_2, x_3, x_4)

Check conditions

Newton's method

Newton's method   From Wikipedia

From Wikipedia, the free encyclopedia

In numerical analysis, **Newton's method**, also known as the **Newton–Raphson method**, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a single-variable function *f* defined for a real variable x, the function's derivative *f* ', and an initial guess x_0 for a root of *f*. If the function satisfies sufficient assumptions and the initial guess is close, then

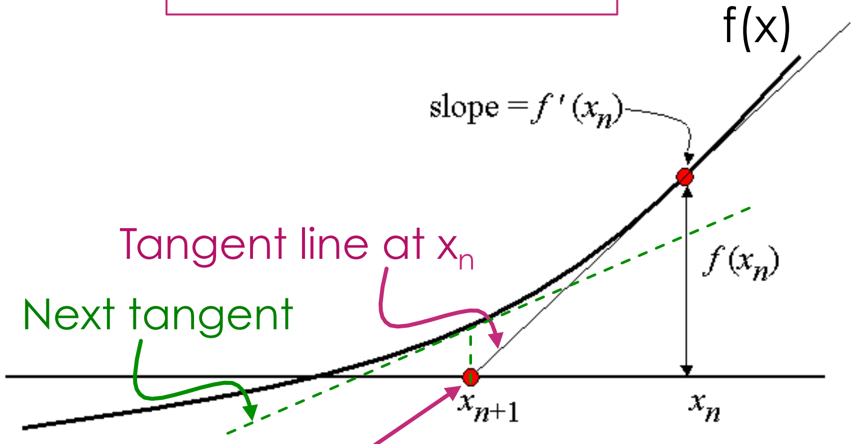
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

is a better approximation of the root than x_0. Geometrically, $(x_1, 0)$ is the intersection of the x-axis and the tangent of the graph of f at $(x_0, f(x_0))$: that is, the improved guess is the unique root of the linear approximation at the initial point. The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently precise value is reached. This algorithm is first in the class of Householder's methods, succeeded by Halley's method. The method can also be extended to complex functions and to systems of equations.

Newton's method



Recursion formula

Next guess   Previous guess

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Derivative at previous guess

Newton's method

- **How to set the derivative?**
  - We can find it numerically
    - Forward estimation ~ **O(h) → order of error**

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

    - Backward estimation ~ **O(h)**

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

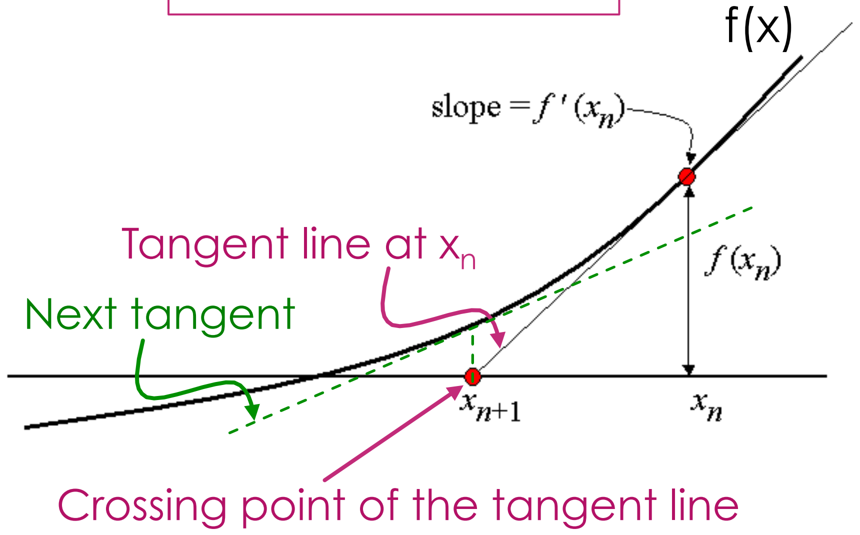
    - Center estimation ~ **O(h²)**

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

  - If we know the analytic function for the derivative, we should be able to set it analytically.

[define a new functional interface]

Newton's method



Recursion formula

Next guess
Previous guess

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

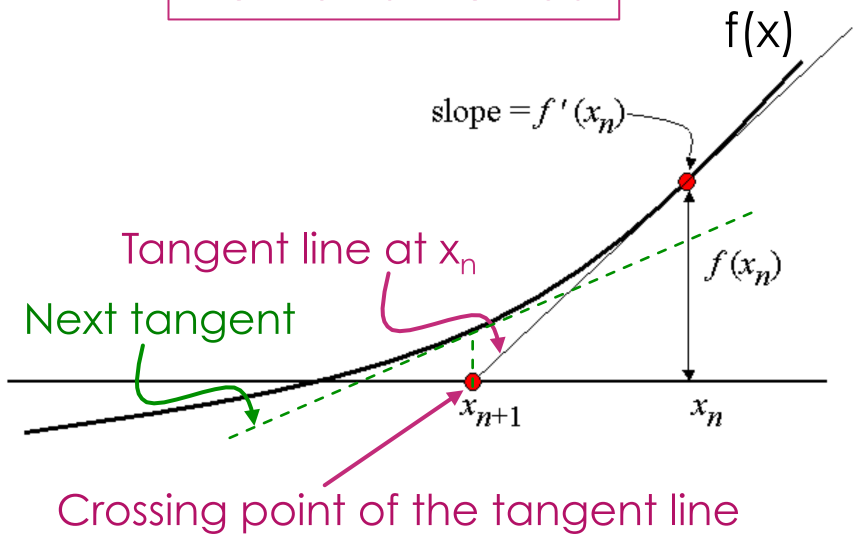
Derivative at previous guess

Newton's method

- **How to find all the roots?**
  - **Step 1**: make sure the roots are bracketed over the search interval
  - **Step 2:** apply Newton's method in each bracket
  - **Step 3:** use a good convergence condition
- **Failures of Newton's method**
  - Bad starting point
    - Derivative is zero right at the beginning
    - Derivative is not defined at the beginning
  - Function is not smooth over the search interval

Unlike bisection, Newton's method can converge even if f(a).f(b)>0

Newton's method



Recursion formula

Next guess

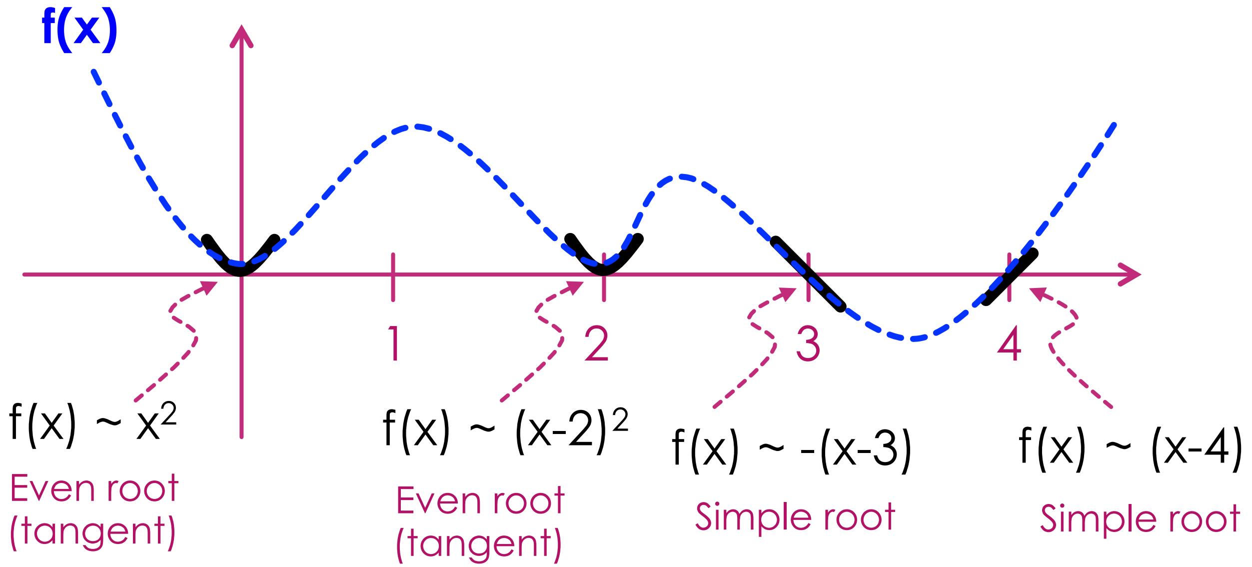
Previous guess

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Derivative at previous guess

Comparison of all methods

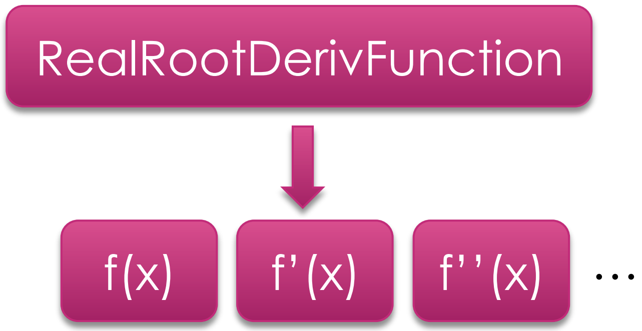
- **Choosing the best method depends on the function**
  - Example: **f(x) = x² (x-2)² (x-3) (x-4)**
  - Any function can be locally expanded as a polynomial (Taylor's theorem)
  - Approximate shape of the function
  - Let's find **all** roots in **(-1, 6)** interval



Root Bracketing is applied

Bisection	➔ (-) Generally fails at even (tangent) roots **(+) Guaranteed** to converge
Trisection	➔ (-) Generally fails at even (tangent) roots **(+) Guaranteed** to converge
Secant	➔ (+) Faster than Bisection **(-) May NOT converge**
Brent	➔ (+) Faster than Secant **(-) May NOT converge**
Newton	➔ (+) Faster than Brent **(-) May NOT converge**

Generalized Taylor’s method



- **Fact 1:** It is easy to find roots of a polynomial
  - Use Laguerre’s Method
- **Fact 2:** Any smooth function can be expanded into a polynomial about a given point
- **Generalized Taylor’s method**:
  - **Step 1**: Find Taylor expansion of f(x) at an initial guess of the root over a given interval
    - Assume that the interval brackets the root
  - **Step 2**: Find the x-intercept of the Taylor polynomial
  - **Step 3**: perform recursion by finding the Taylor expansion at the new x-intercept

Degree = 1 → Newton’s method

- Finding higher order derivatives numerically may be challenging

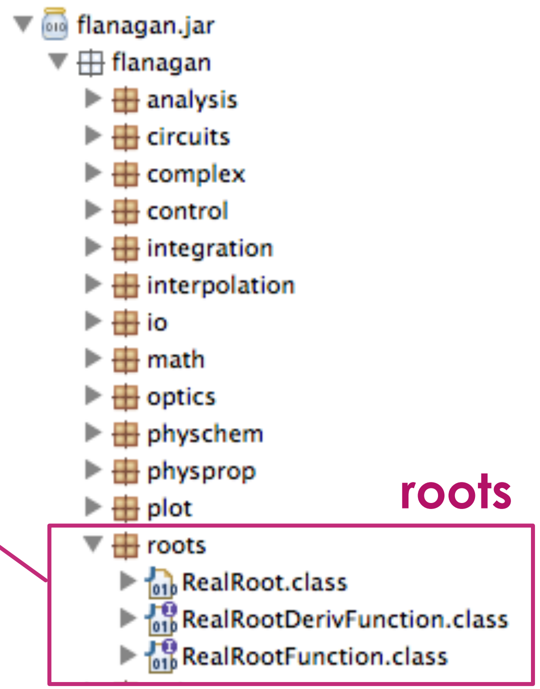
1) **RealRootDerivFunction**

2) **Symbolic** Functions

Other Root-Finding Libraries

- **Michael Flanagan's** java scientific library
  - https://www.ee.ucl.ac.uk/~mflanaga/java/
  - Root package is very similar to what we developed
    - RealRootFunction interface
      - ✓ Defines real-valued function f(x)
    - RealRootDerivFunction interface
      - ✓ Defines f(x) and its derivatives
    - RealRoot class
      - ✓ Implements the root-finding algorithms
        - Bisection
        - Brent
        - Newton-Raphson
        - **False Position** method (improved secant method)
        - Also allows for automatic extension of the original interval

Flanagan's library



[**a**, **b**] → dx = (b-a)/2

Extension of lower bound

Extension of upper bound

[**a-dx**, **b+dx**]