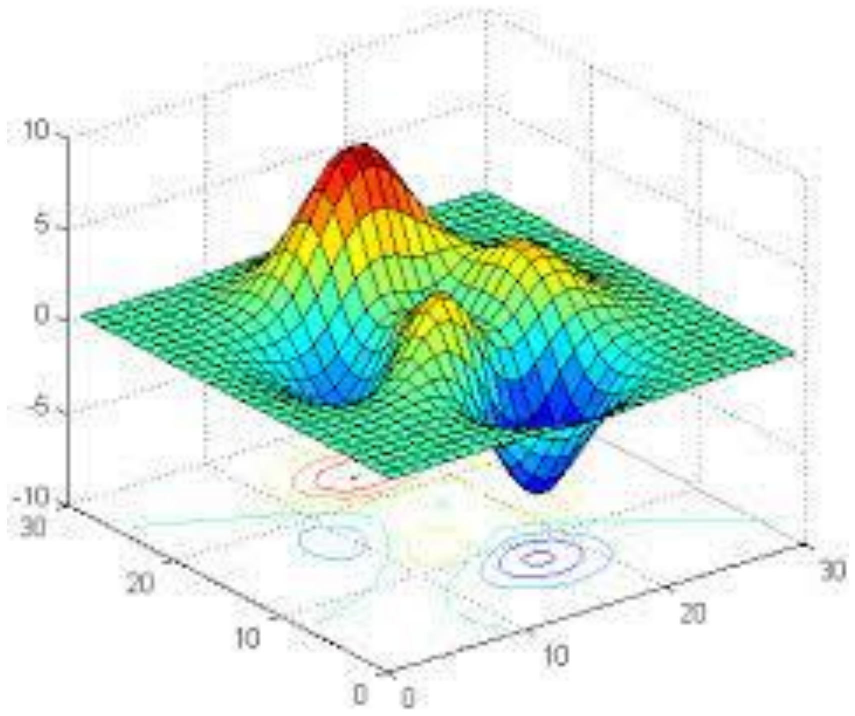
Khemis Miliana University
Faculty of Sciences of Matter and Computer Science
Department of Physics

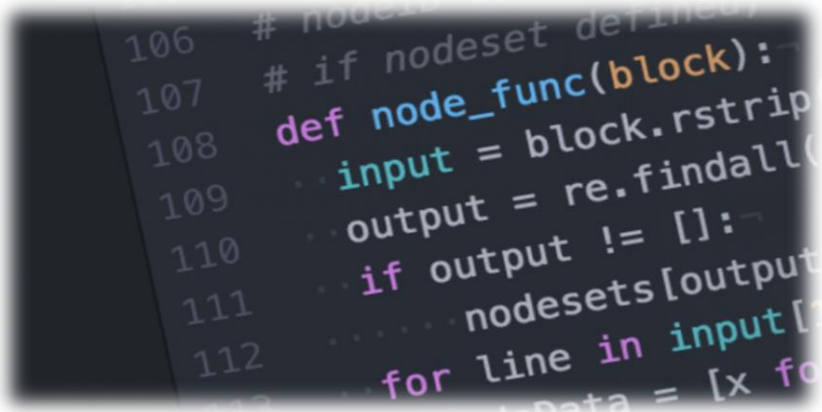


Numerical Methods & Scientific Programming

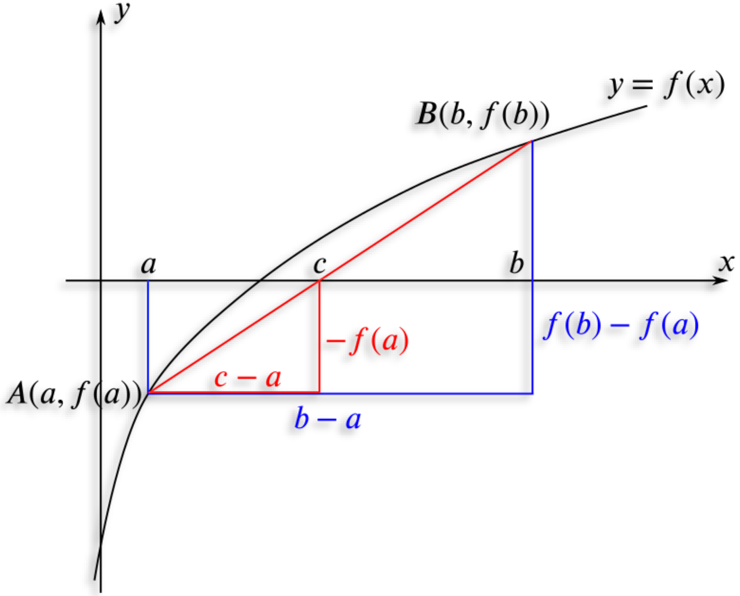
Dr. Salah-Eddine BENTRIDI
s.bentridi@univ-dbkm.dz
Univ. Khemis-Miliana

Content of the program

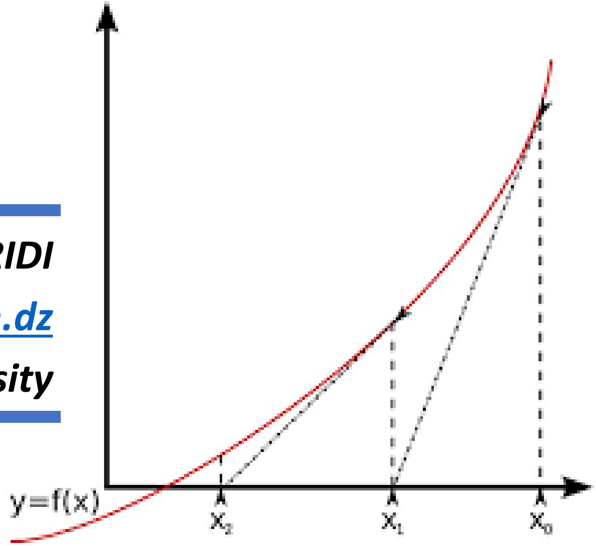
- Chapter 01: ***Initiation to a programming language (Python)***
  - ***Hands-on-Pythn; Basics of Python***
- Chapter 02: ***Numerical Integration***
  - ***Trapezoidal rule; Simpson’s method***
- Chapter 03: ***Numerical Solution of equations – Root finding***
  - ***Bisection method; Newton’s Method***
- Chapter 04: ***Numerical resolution of differential equations***
  - ***Euler’s method; Runge-Kutta method***
- Chapter 05: ***Numerical resolution of linear equations system***
  - ***Gauss method, Gauss-Seidel method***



Chapter 03: Numerical Solution of equations – Root finding



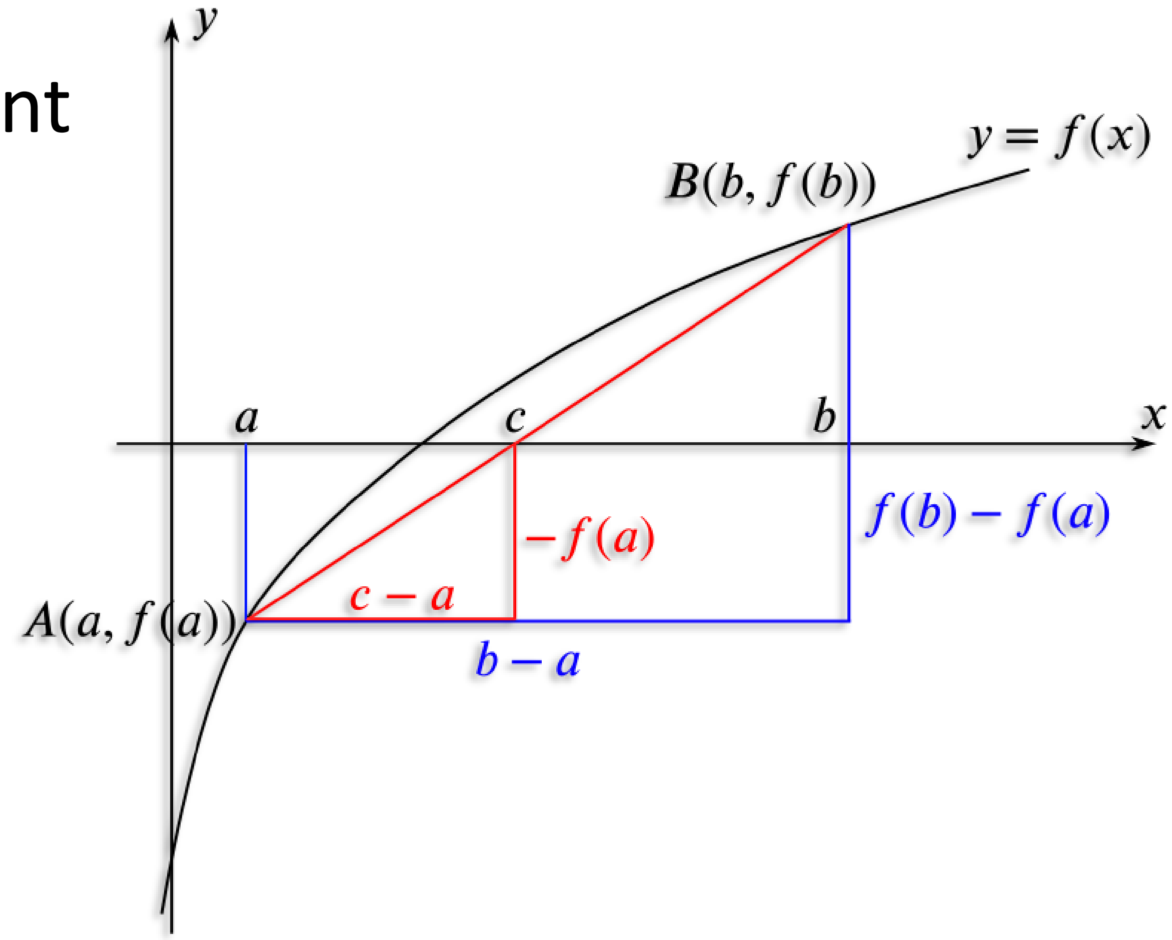
Bisection method and Newton’s Method



Dr. Salah-Eddine BENTRIDI
s.bentridi@univ-dbkm.dz
Khemis-Miliana University

Outline

- Root finding problem statement
- Tolerance
- Bisection Method
- Newton-Raphson Method
- Root finding using Scipy



I. Root Finding Problem Statement

The root or zero of a function, $f(x)$ *, is an* x_r *such that* $f(x_r) = 0$*.*

For functions such as $f(x) = x2 - 9$ *, the roots are clearly* 3 *and* -3*.*

However, for other functions such as $h(x) = cos(x) - x$*, determining an analytic, or exact, solution for the roots of functions can be difficult.*

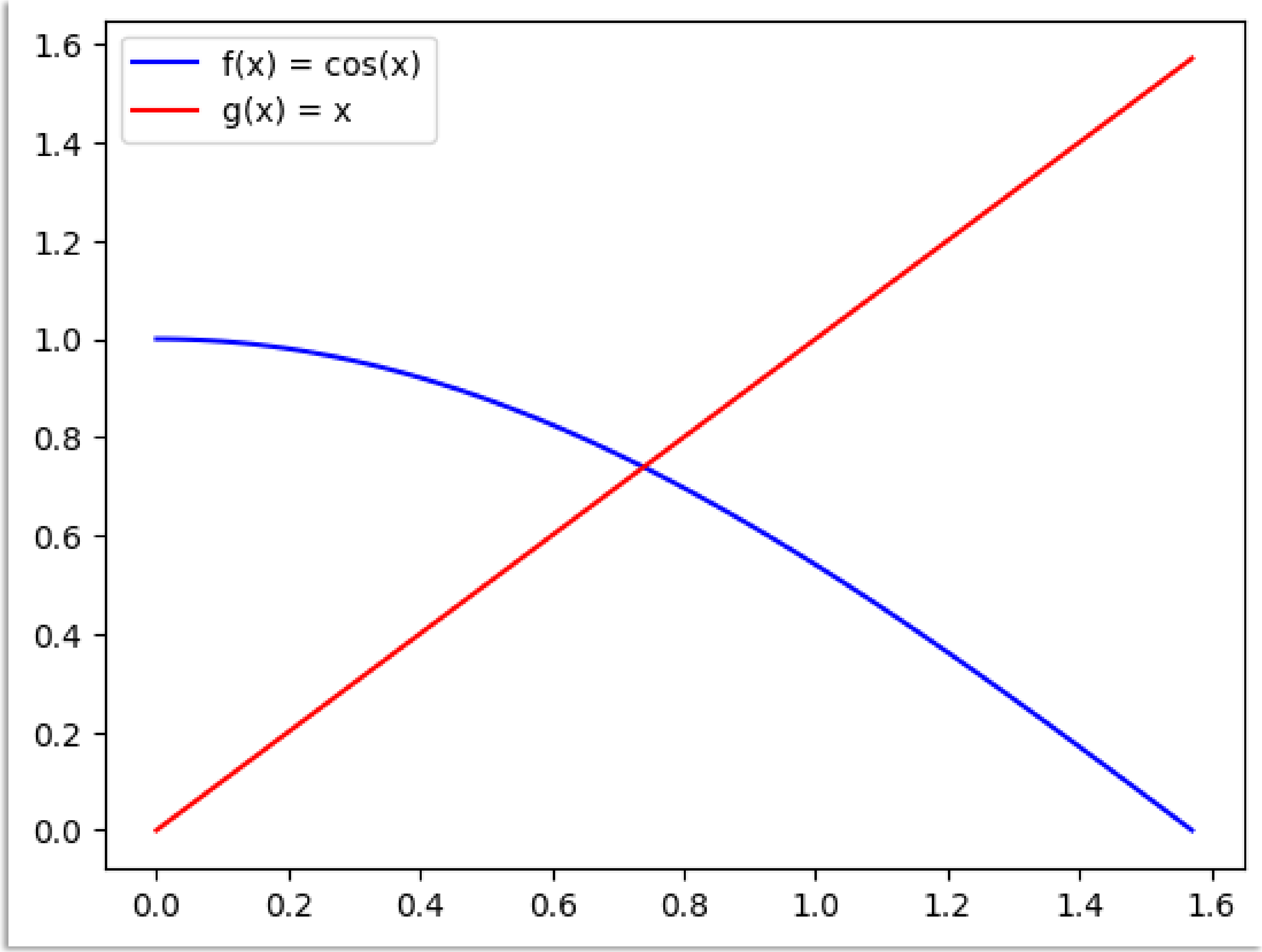
For these cases, it is useful to generate numerical approximations of the roots of f *and understand the limitations in doing so.*

Let's try to find the root of equation $h(x)$ *cited above, by plotting both elementary functions:* $y_1 = cos(x)$ *and* $y_2 = x$*.*

I. Root Finding Problem Statement

Let's try to find the root of equation $h(x)$ cited above, by plotting both elementary functions: $y_1 = cos(x)$ and $y_2 = x$.

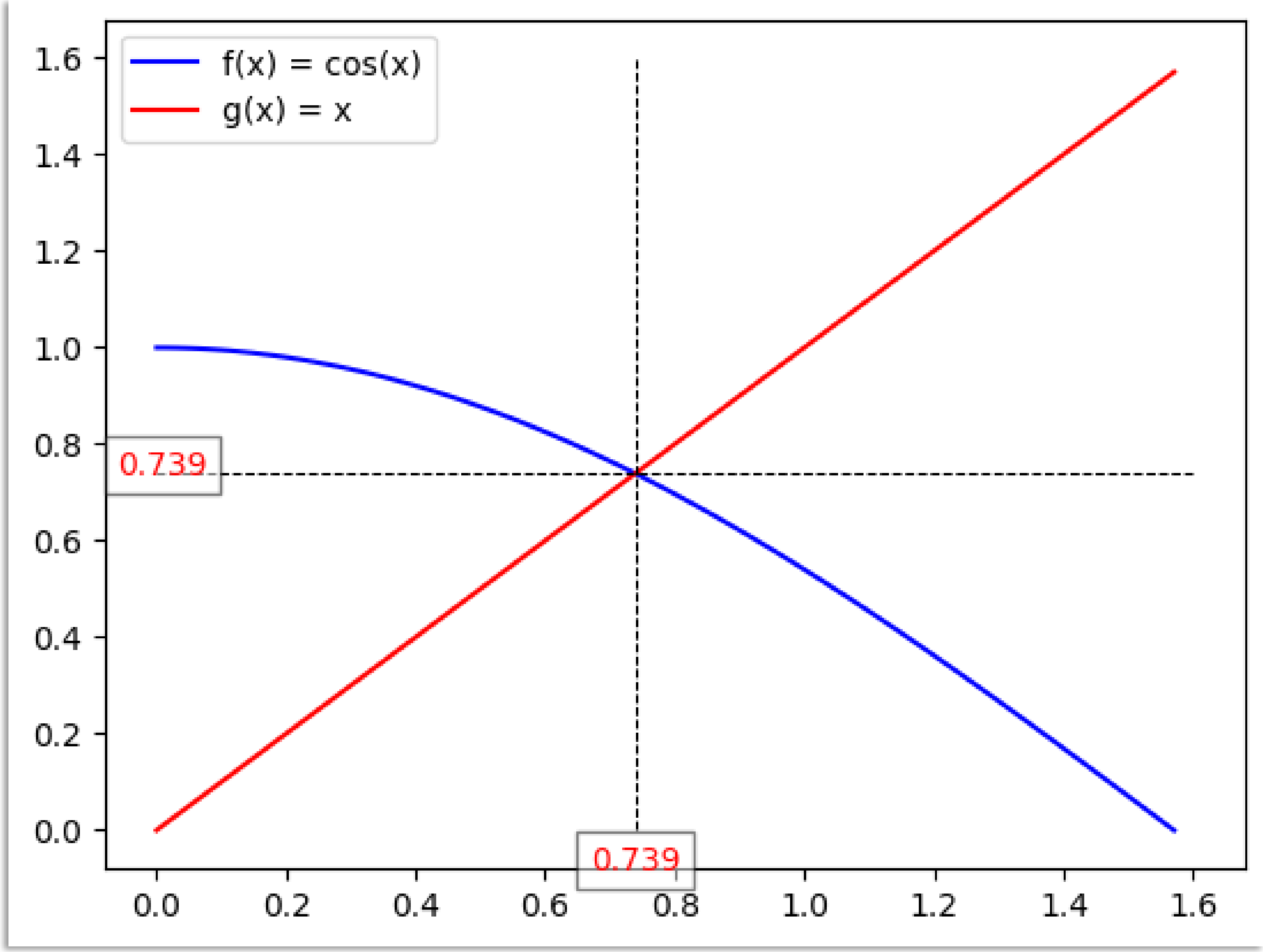
1. *Make a guess about the interval where we can find the solution.*
2. *Use your calculator to move on this interval*



I. Root Finding Problem Statement

Let's try to find the root of equation $h(x)$ cited above, by plotting both elementary functions: $y_1 = cos(x)$ and $y_2 = x$.

1. ***Make a guess about the interval where we can find the solution.***
2. ***Use your calculator to move on this interval***



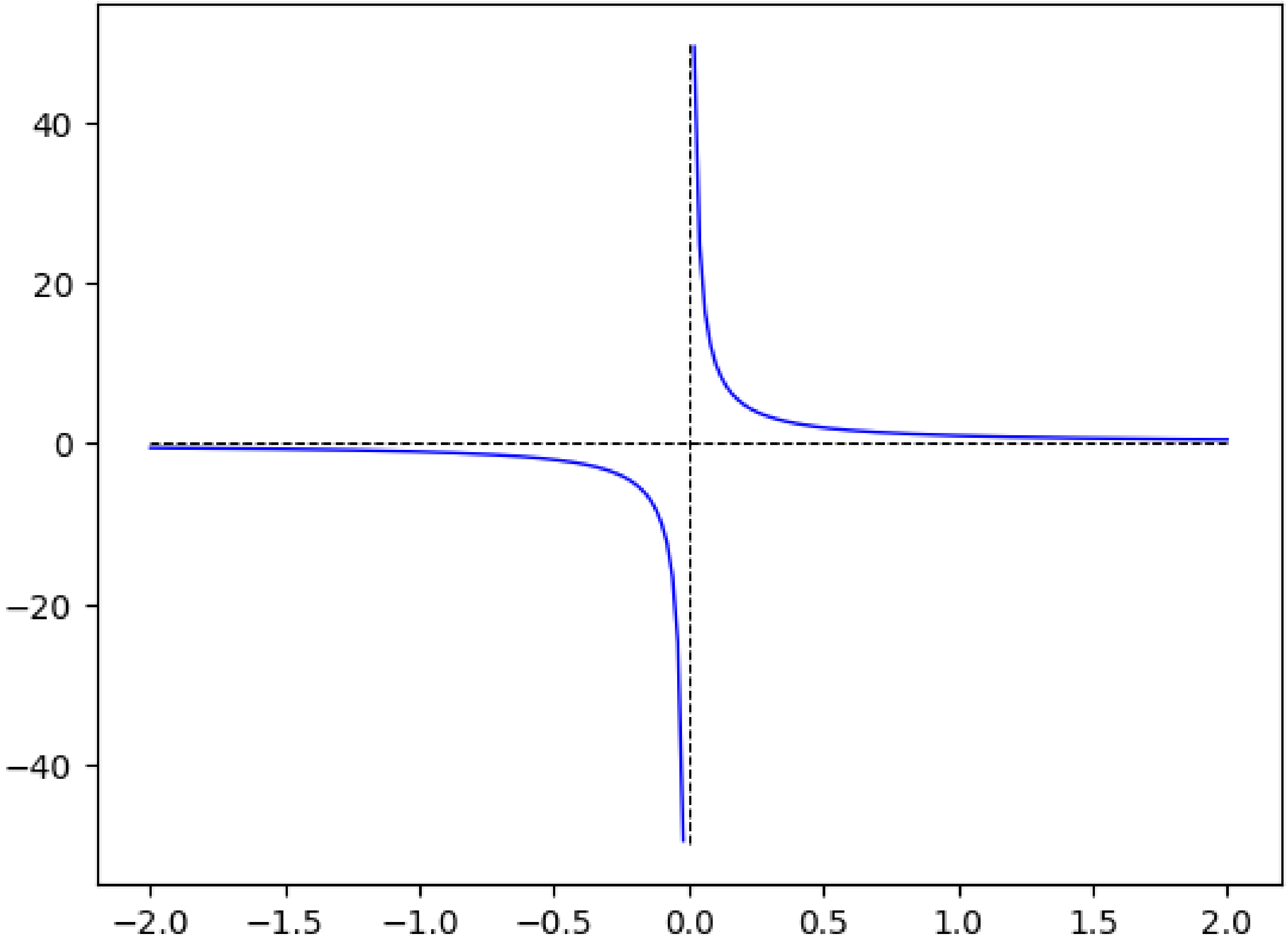
I. Root Finding Problem Statement

It is important to verify that analyzed function will accept a root or not!!! Take the following example:

$$f(x) = \frac{1}{x}$$

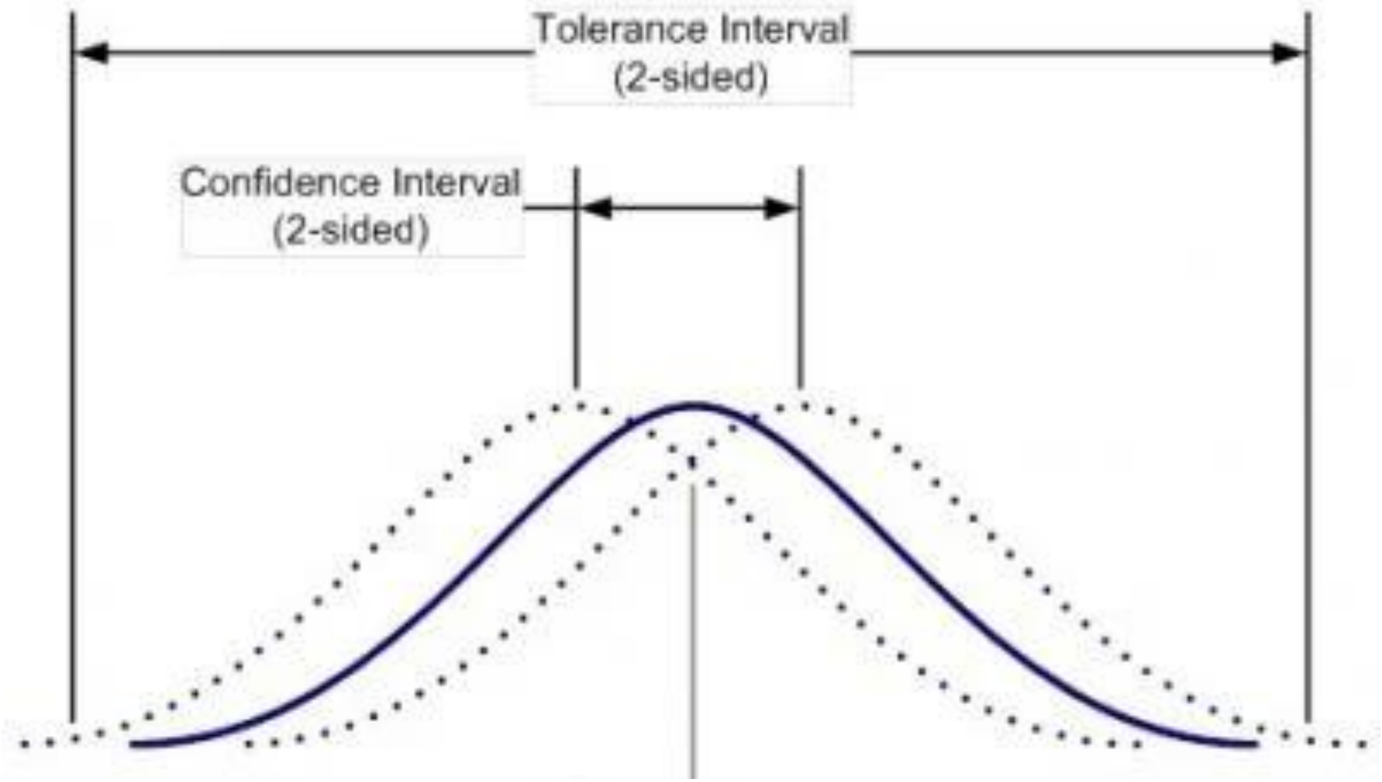
Which vanishes when $x \to \pm\infty$, but does not have a finite root x_r for which:

$$f(x_r) = 0$$



II. Tolerance

- ***In engineering and science, error is a deviation from an expected or computed value.***
- ***Tolerance is the level of error that is acceptable for an engineering application. We say that a computer program has converged to a solution when it has found a solution with an error smaller than the tolerance.***
- **When computing roots numerically, or conducting any other kind of numerical analysis, it is important to establish both a metric for error and a tolerance that is suitable for a given engineering/science application.**



II. Tolerance

- *For computing roots, we want an x_r such that $f(x_r)$ is very close to $\mathbf{0}$. Therefore $|f(x)|$ is a possible choice for the measure of error since the smaller it is, the likelier we are to a root.*
- *Also, if we assume that x_i is the i^{th} guess of an algorithm for finding a root, then: $|xi + 1 - xi|$ is another possible choice for measuring error, since we expect the improvements between subsequent guesses to diminish as it approaches a solution.*
- *But one should use this concept carefully, because these different choices have their advantages and disadvantages.*

Case 01: Let error be measured by $e = |f(x)|$ and tol be the acceptable level of error. The function $f(x) = x^2 + tol/2$ has no real roots. However, $|f(0)| = tol/2$ is therefore acceptable as a solution for a root finding program.

II. Tolerance

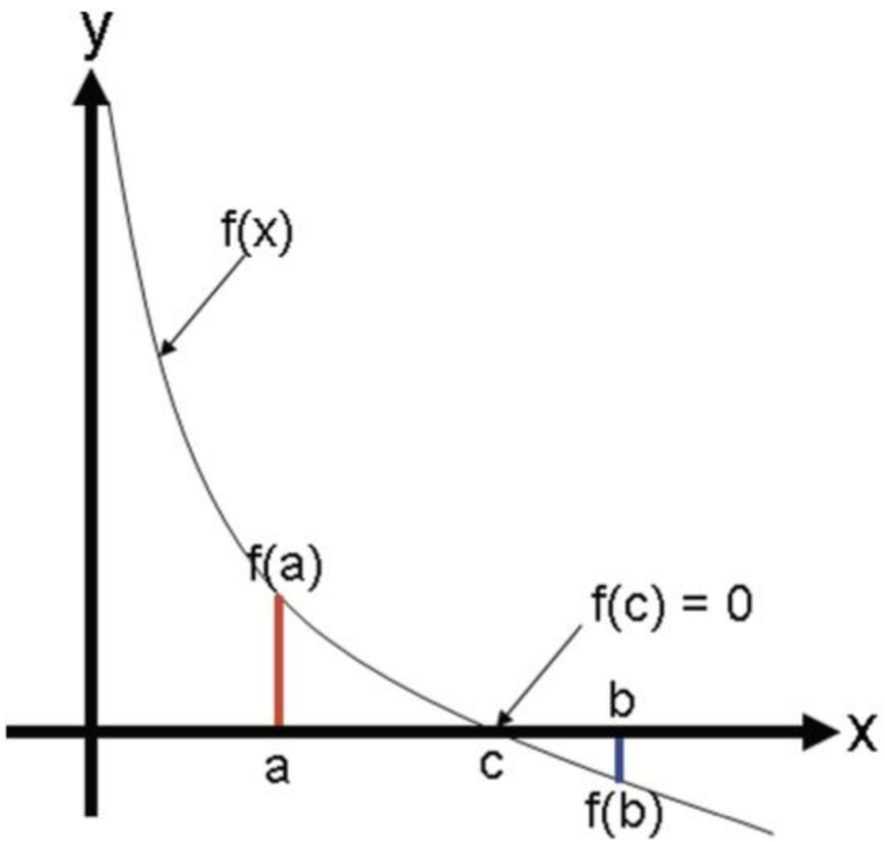
- *For computing roots, we want an* x_r *such that* $f(x_r)$ *is very close to* 0. *Therefore* $|f(x)|$ *is a possible choice for the measure of error since the smaller it is, the likelier we are to a root.*
- *Also, if we assume that* x_i *is the* i^{th} *guess of an algorithm for finding a root, then:* $|xi + 1 - xi|$ *is another possible choice for measuring error, since we expect the improvements between subsequent guesses to diminish as it approaches a solution.*
- *As will be demonstrated in the following examples, these different choices have their advantages and disadvantages.*

Case 02: Let error be measured by $e = |x_{i+1} - x_i|$ *and* tol *be the acceptable level of error. The function* $f(x) = 1/x$ *has no real roots, but the guesses* $x_i = -\frac{tol}{4}$ *and* $x_{i+1} = \frac{tol}{4}$ *have an error of* $e = tol/2$ *and is an acceptable solution for a computer program.*

III. Bisection method

Intermediate Value Theorem

- *The Intermediate Value Theorem says that if $f(x)$ is a [continuous function](#) between a and b, and $sign(f(a)) \neq sign(f(b))$, then there must be a c, such that $a < c < b$ and $f(c) = 0$.*

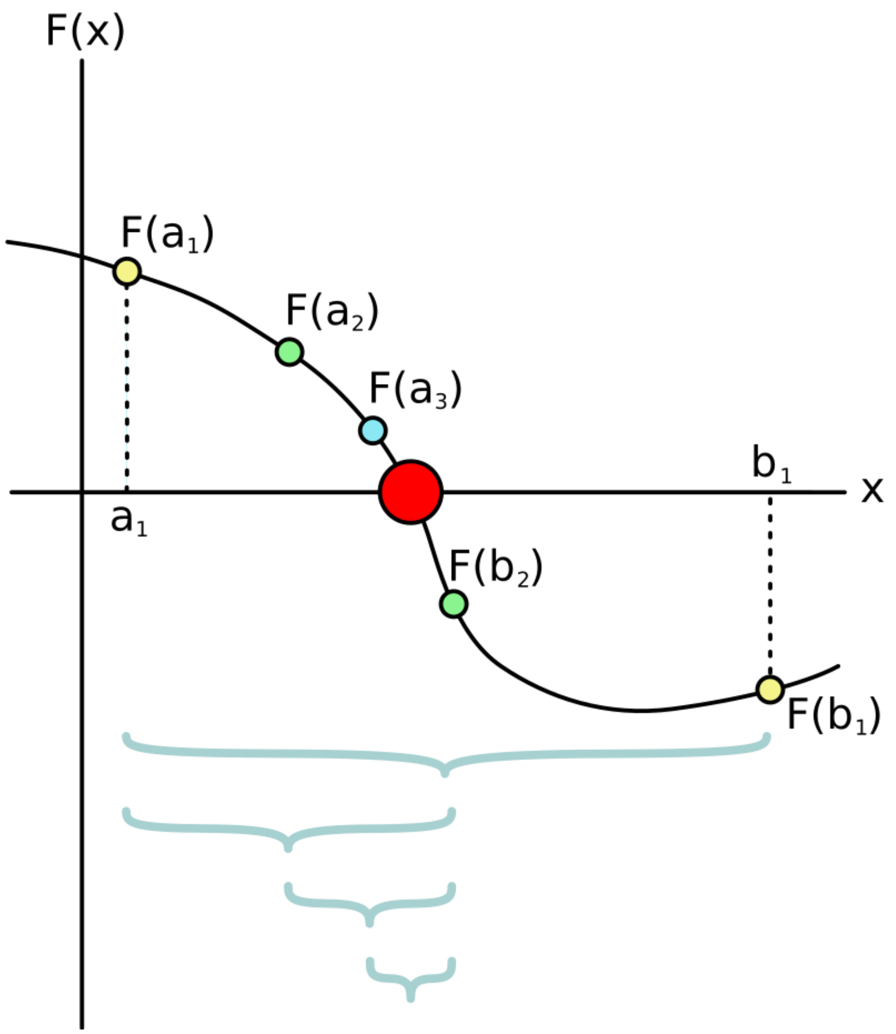


III. Bisection method

- *The bisection method uses the intermediate value theorem iteratively to find roots. Let* $f(x)$ *be a continuous function, and a and b be real scalar values such that* $a < b$.
- *Assume, without loss of generality, that:*

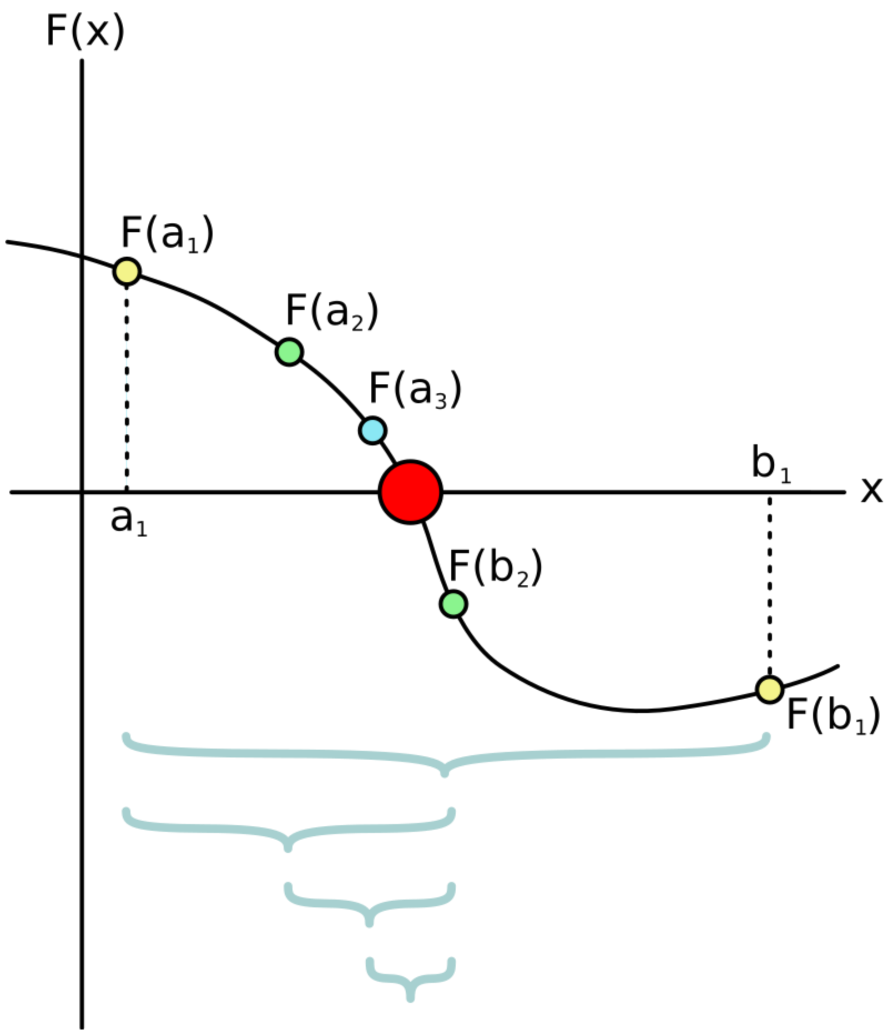
$$f(a) > 0 \text{ and } f(b) < 0.$$

- *Then by the intermediate value theorem, there must be a root on the open interval* $[a, b]$. *Now let* $m = (a + b)/2$, *the midpoint between* a *and* b.



III. Bisection method

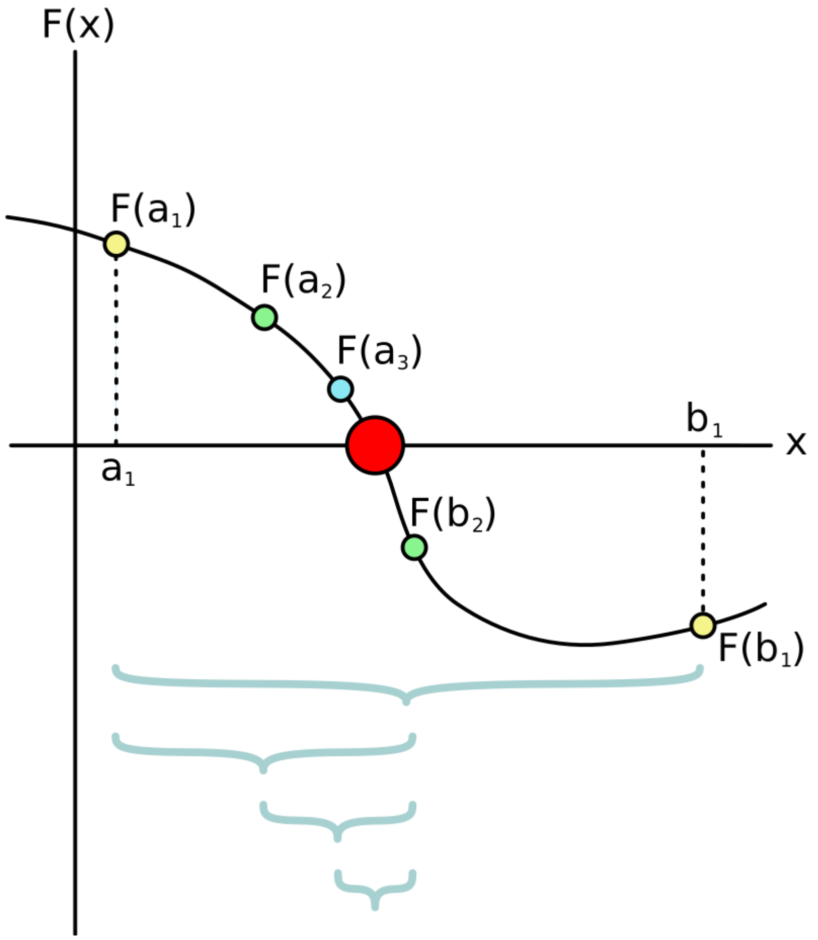
- *If $f(m) = 0$ or is close enough, then m is a root.*
- *If $f(m) > 0$, then m is an improvement on the left bound, a, and there is guaranteed to be a root on the open interval $[m, b]$*
- *If $f(m) < 0$, then m is an improvement on the right bound, b, and there is guaranteed to be a root on the open interval $[a, m]$.*



III. Bisection method

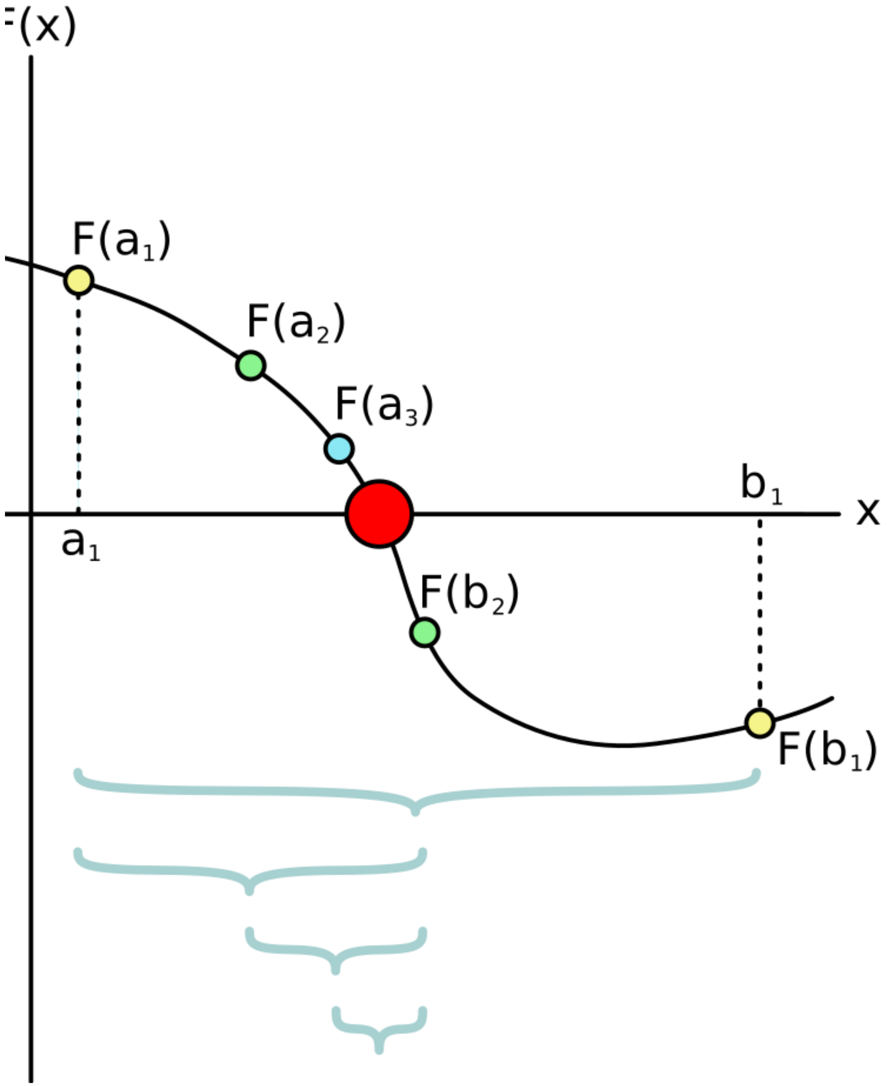
Algorithm Bisection method

Set a tolerance TOL for accuracy
Initialize the interval $[a, b]$
Set $k = 0$
Check if $f(a) \cdot f(b) < 0$ (if not then quit)
while $|b - a| > TOL$ **do**
    Set $x_k = (a + b)/2$ (the new approximation)
    Check in which interval $[a, x_k]$ or $[x_k, b]$ the function changes sign
    if $f(x_k) \cdot f(a) < 0$ **then**
        $b = x_k$
    else if $f(x_k) \cdot f(b) < 0$ **then**
        $a = x_k$
    else
        Break the loop because $f(x_k) = 0$
    end if
    $k = k + 1$
end while
Return the approximation of the root x^*



III. Bisection method

```python
def my_bisection(f, a, b, tol):
    # check if a and b bound a root
    if np.sign(f(a)) == np.sign(f(b)):
        raise Exception(
         "The scalars a and b do not bound a root")
    # get midpoint
    m = (a + b)/2
    if np.abs(f(m)) < tol:
        # stopping condition, report m as root
        return m
    elif np.sign(f(a)) == np.sign(f(m)):
        # case where m is an improvement on a.
        # Make recursive call with a = m
        return my_bisection(f, m, b, tol)
    elif np.sign(f(b)) == np.sign(f(m)):
        # case where m is an improvement on b.
        # Make recursive call with b = m
        return my_bisection(f, a, m, tol)
```



IV. Newton-Raphson Method

- *Let* $f(x)$ *be a smooth and continuous function and* x_r *be an unknown root of* $f(x)$*.*
- *Now assume that* x_0 *is a guess for* x_r*. Unless* x_0 *is a very lucky guess,* $f(x_0)$ *will not be a root.*
- *Given this scenario, we want to find an* x_1 *that is an improvement on* x_0 *(i.e., closer to* x_r *than* x_0 *.*
- *If we assume that* x_0 *is "close enough" to* x_r*, then we can improve upon it by taking the linear approximation of* $f(x)$ *around* x_r*, which is a line, and finding the intersection of this line with the* $x - axis$*.*
- *Written out, the linear approximation of* $f(x)$ *around* x_0 *is:*

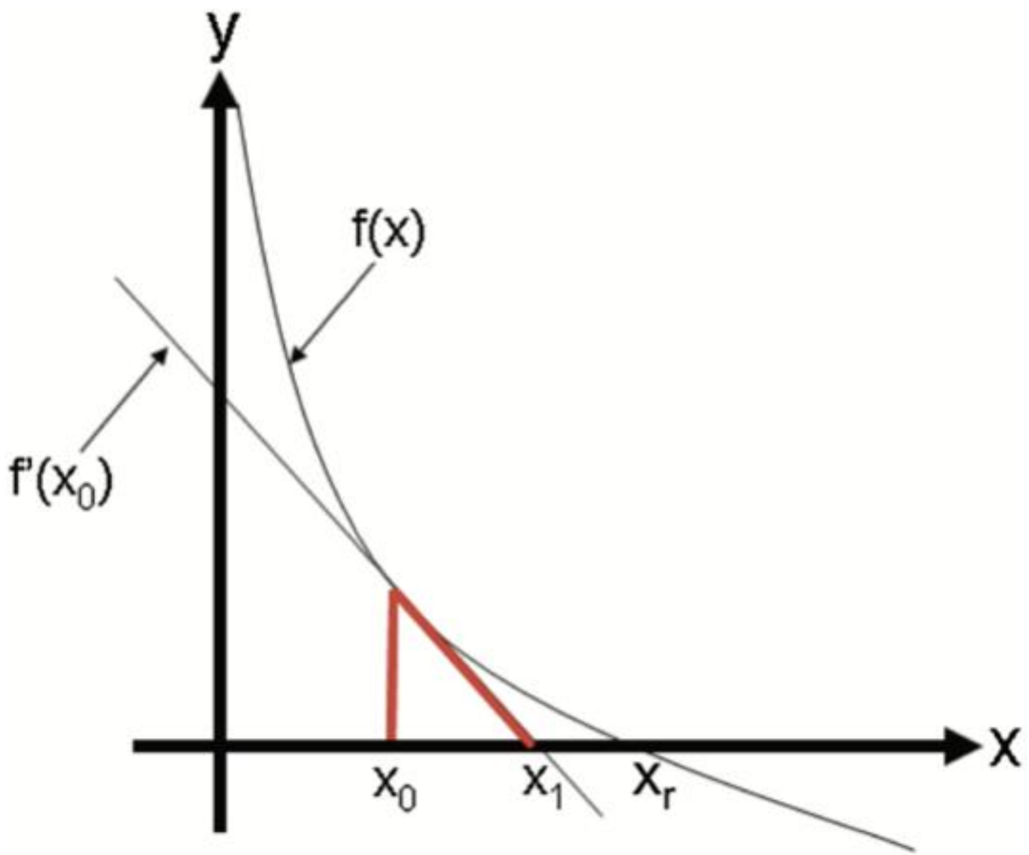
$$f(x) \approx f(x_0) + f'(x_0)(x - x_0).$$

IV. Newton-Raphson Method

- ***Using this approximation, we find x_1 such that $f(x_1) = 0$.***
- ***Plugging these values into the linear approximation results in the equation:***

$$0 = f(x_0) + f'(x_0)(x_1 - x_0)$$

which when solved for x_1 is $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

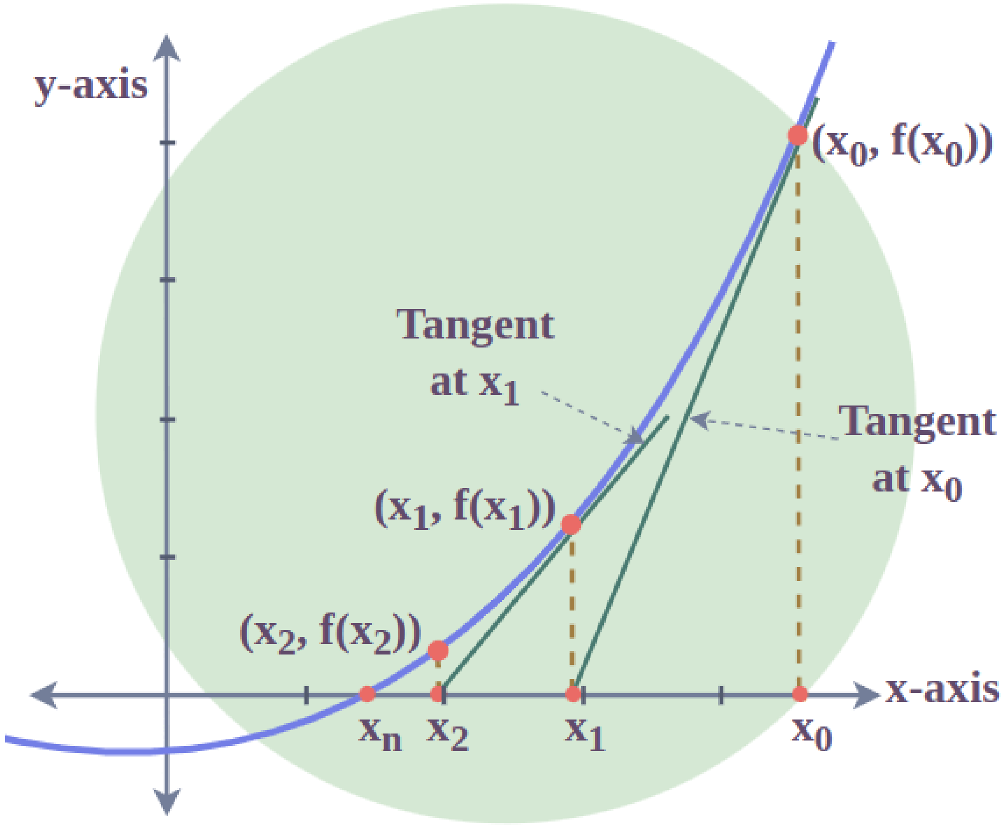


IV. Newton-Raphson Method

- ***Written generally, a Newton step computes an improved guess, x_i, using a previous guess x_{i-1}, and is given by the equation***

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

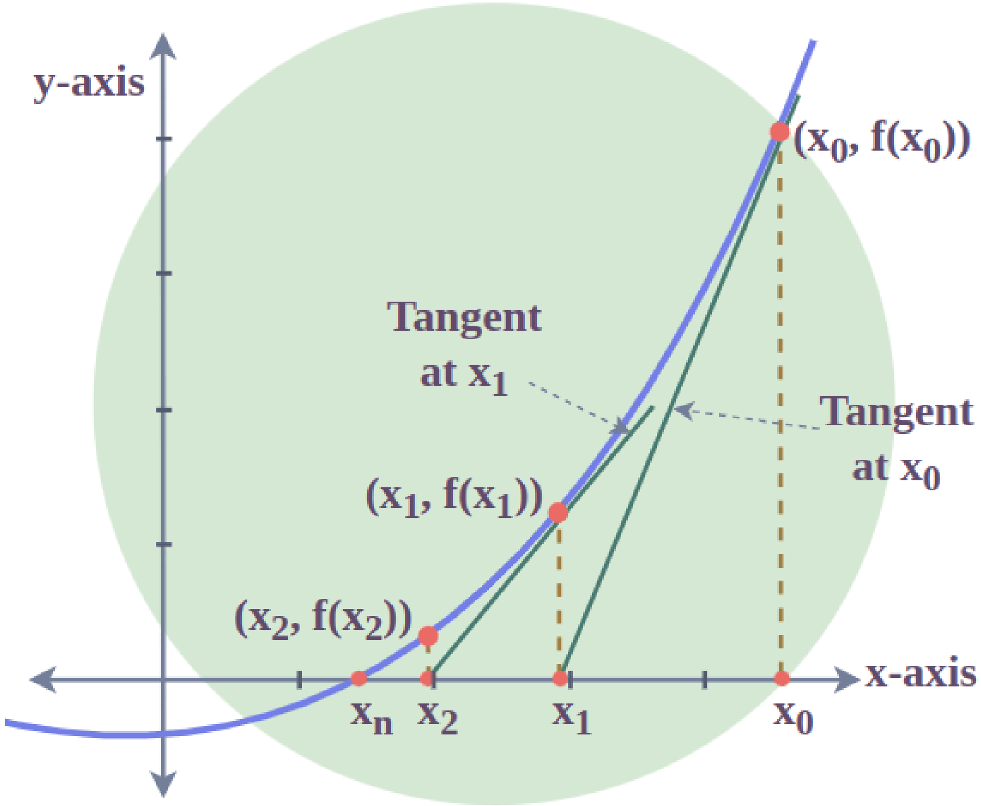
- **The Newton-Raphson Method of finding roots iterates Newton steps from x_0 until the error is less than the tolerance.**



IV. Newton-Raphson Method

Algorithm Newton's method

Set a tolerance TOL for the accuracy
Set the maximum number of iterations $MAXIT$
Set $k = 0$
Initialize x_k and $Error = TOL + 1$
while $Error > TOL$ and $k < MAXIT$ **do**
  if $f'(x_k) \neq 0$ **then**
    Compute $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
    $Error = |x_{k+1} - x_k|$
    Increase the counter $k = k + 1$
  end if
end while
Return the approximation of the root x^*



IV. Newton-Raphson Method

```python
def my_newton(f, df, x0, tol):
    # output is an estimation of the root of f
    # using the Newton Raphson method
    # recursive implementation
    if abs(f(x0)) < tol:
        return x0
    else:
        return my_newton(f, df, x0 - f(x0)/df(x0), tol)
```