Computational Physics

Solution to Nonlinear Equations

Prof. Paul Eugenio
Department of Physics
Florida State University
Mar. 12, 2019

http://hadron.physics.fsu.edu/ ~eugenio/comphy

Assigned Reading

- **Read Chapter 6 Section 1-4:**
  - Solution of Linear Equations
  - Solution of Nonlinear Equations
- Turn-In Questions
  - Two questions on material due Mar 26

Solution of Nonlinear Equations

Finding the zeros of a function!

- Relaxation Method
- Binary Search
- False Position Method
- Newton-Raphson Method
- Secant Method

Finding Zeros of Functions

One of the Most Basic Tasks:

Solving Equations Numerically

- **F**($\mathbf{x}_n$) = 0 N-Dimensional Case
  - Generic
    - N-Equations – N-solutions
    - Distinct, Point-like, Separated
  - Non-Generic
    - Degenerate
      - Continuous family of solutions
    - Nonlinear
      - May have no real solution
- $f(x) = 0$ One Dimensional Case
  - Possible to trap a root between bracketing values, and then hunt it down.

Relaxation Method

- Simple iteration of the equation
  - Drawbacks
    - equation must be of the form `x = f(x)`
    - method often does not converge
    - method has difficulties finding multiple solutions

Basic Approach

1) Guess an initial value x_i. Also choose a solution accuracy.
2) Calculate the value of f(x)
3) if $| f(x) - x_i | > \text{accuracy}$
   - set x_i to f(x)
   - repeat from step 2

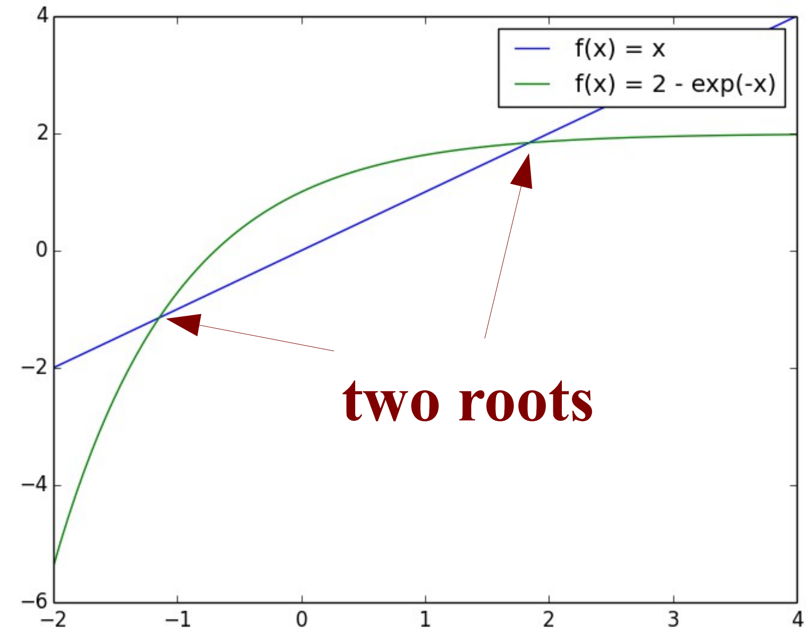
Example: $x = 2 - e^{-x}$

Relaxation Method

```
import numpy as np

target = 1e-10
x = 1
xold = float("inf")
while np.abs(x-xold)> target
    xold = x
    x = 2 - np.exp(-x)
    print(x)
```

```
hpc-login 663% relaxation.py
1.63212055883
1.80448546585
1.83544089392
1.84045685534
1.84125511391
1.84138178281
1.84140187354
1.84140505985
1.84140556519
1.84140564533
1.84140565804
1.84140566006
1.84140566038
1.84140566043
```

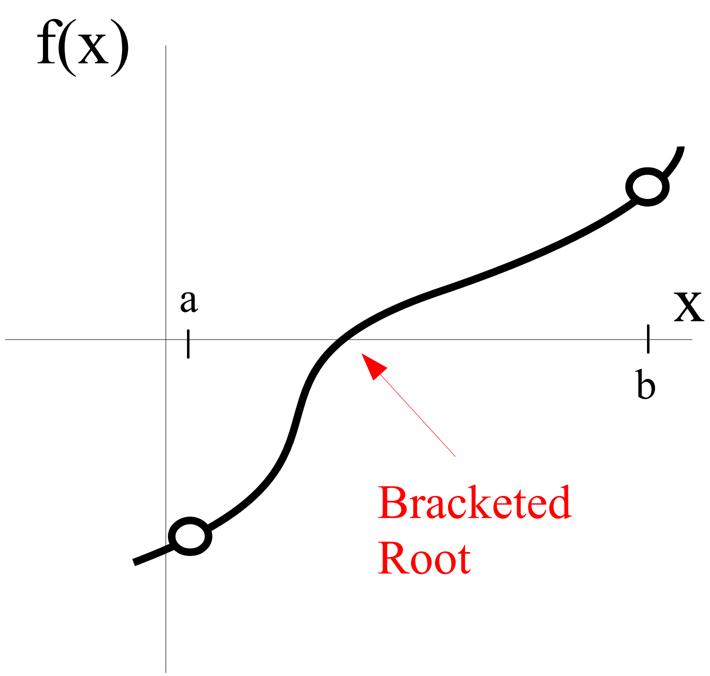


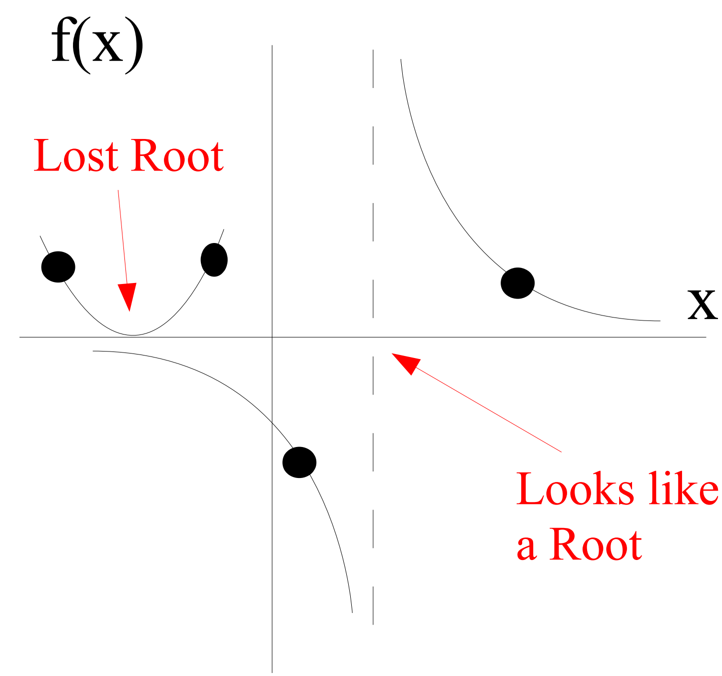
Sometimes changing the initial guess can results in finding another root, <u>but in this case only one root is found independent of the initial value!</u>

Bracketing & Bisection
(Binary Search)

Finding roots of f(x) = 0

- If in {a,b}, f(a) & f(b) have opposite signs and f(x) is continuous then (at least) one root must exist.



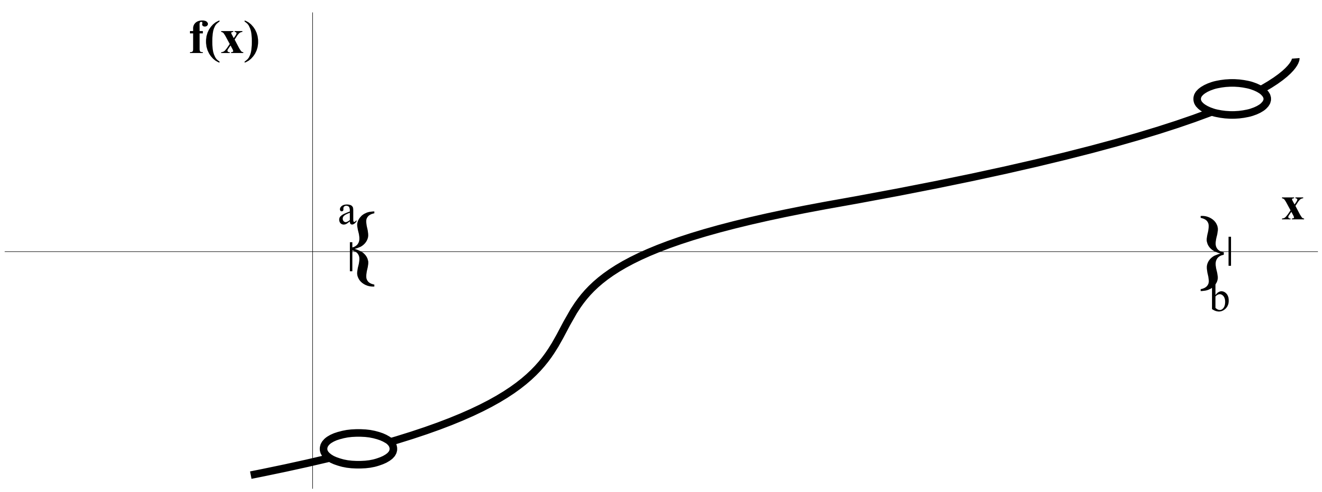


Does not always work

Bracketing & Bisection

Basic Approach

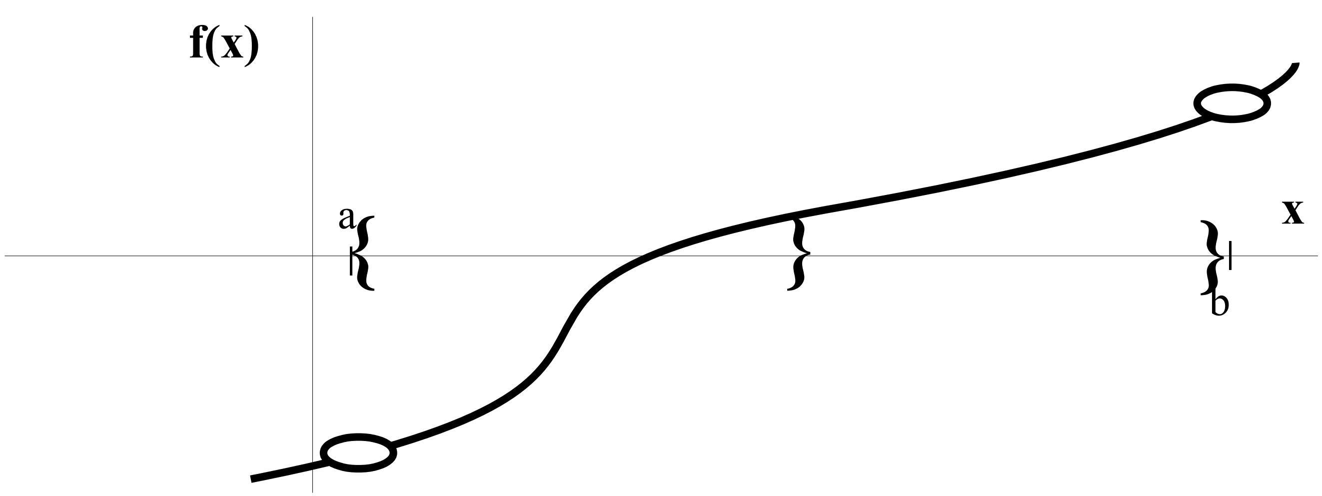
1) Given x_a,x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the value of the midpoint $x_m = ½ (x_a + x_b)$
3)
   a) if $f(x_m) = 0$ stop
   b) if $f(x_m)*f(x_a) > 0$ replace x_a with value of x_m
   c) else replace x_b with value of x_m
4) if $|x_a - x_b|$ > accuracy, repeat from step 2



Bracketing & Bisection

Basic Approach

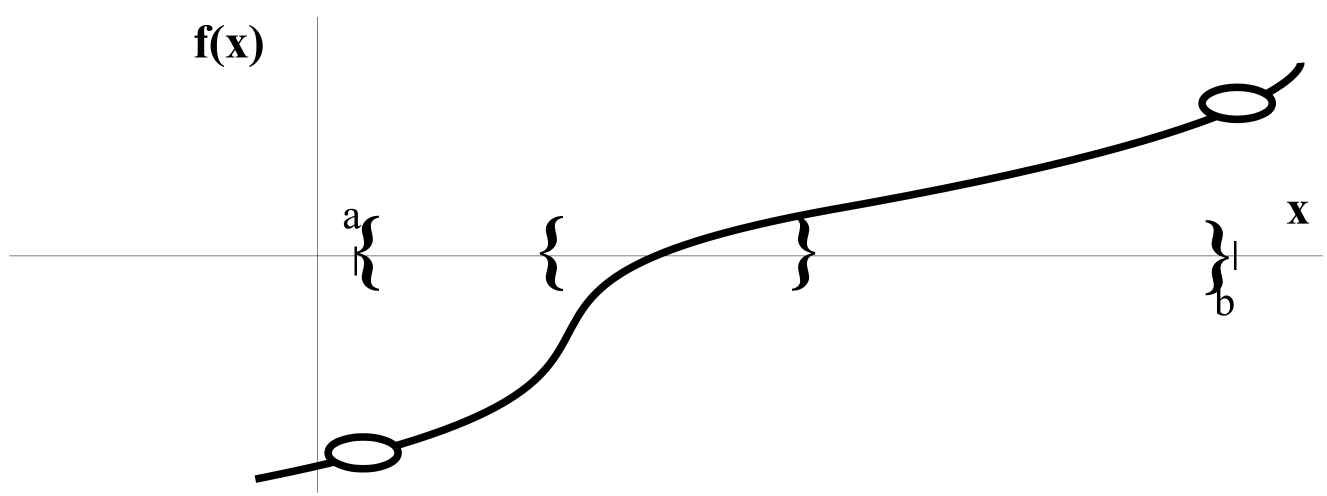
1) Given x_a, x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the value of the midpoint $x_m = ½ (x_a + x_b)$
3)
   a) if $f(x_m) = 0$ stop
   b) if $f(x_m)*f(x_a) > 0$ replace x_a with value of x_m
   c) else replace x_b with value of x_m
4) if $|x_a - x_b| >$ accuracy, repeat from step 2



Bracketing & Bisection

Basic Approach

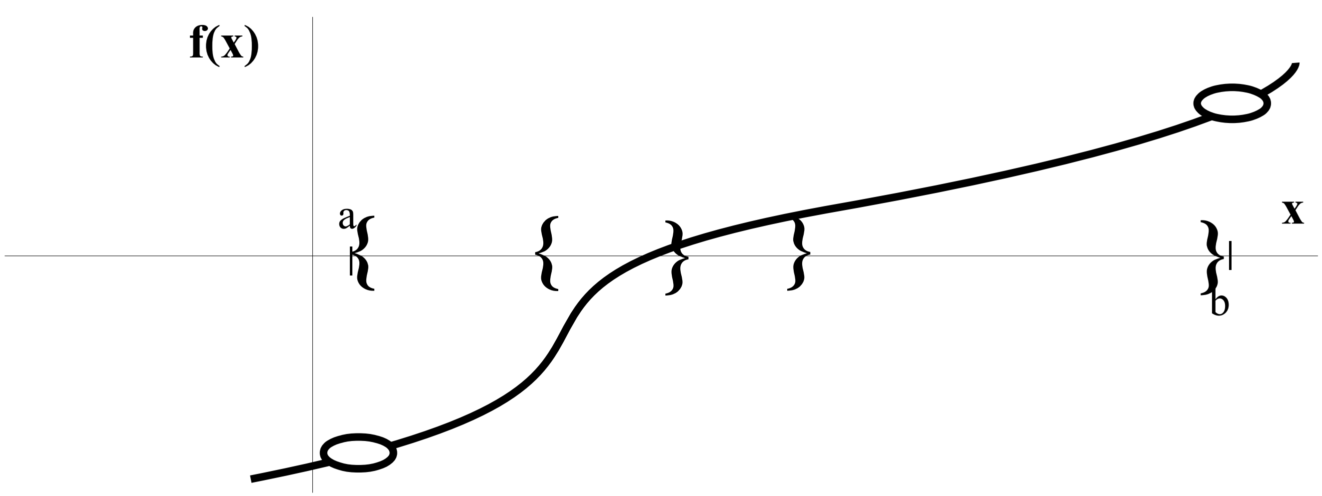
1) Given x_a,x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the value of the midpoint $x_m = ½ (x_a + x_b)$
3)
   a) if $f(x_m) = 0$ stop
   b) if $f(x_m)*f(x_a) > 0$ replace x_a with value of x_m
   c) else replace x_b with value of x_m
4) if $|x_a - x_b|$ > accuracy, repeat from step 2



Bracketing & Bisection

Basic Approach

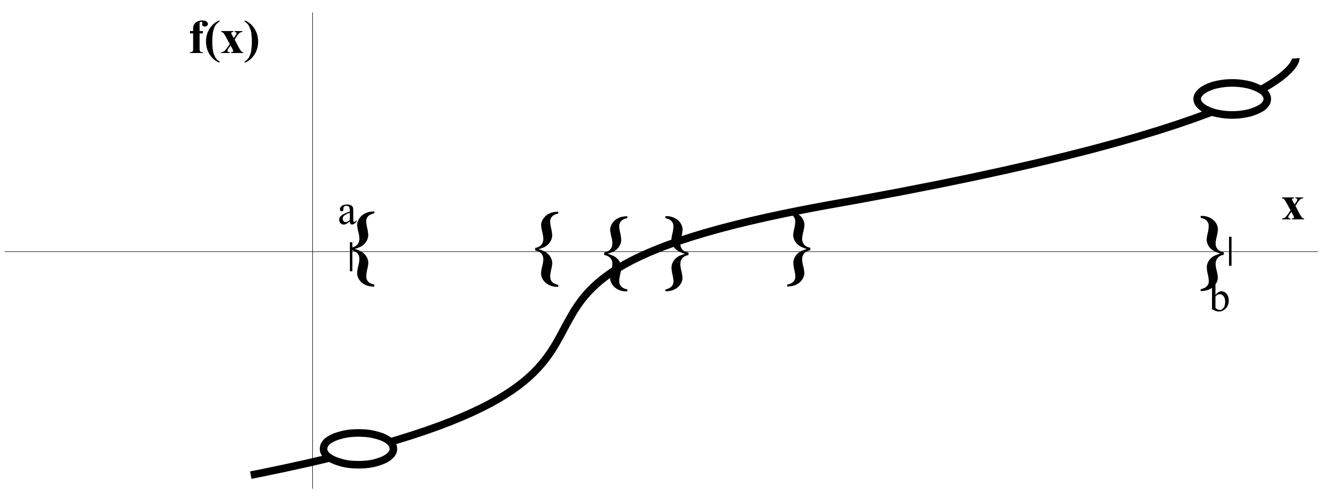
1) Given x_a,x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the value of the midpoint $x_m = ½ (x_a + x_b)$
3)
   a) if $f(x_m) = 0$ stop
   b) if $f(x_m)*f(x_a) > 0$ replace x_a with value of x_m
   c) else replace x_b with value of x_m
4) if $|x_a - x_b|$ > accuracy, repeat from step 2



Bracketing & Bisection

Basic Approach

1) Given x_a,x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the value of the midpoint $x_m = ½ (x_a + x_b)$
3)
   a) if $f(x_m) = 0$ stop
   b) if $f(x_m)*f(x_a) > 0$ replace x_a with value of x_m
   c) else replace x_b with value of x_m
4) if $|x_a - x_b|$ > accuracy, repeat from step 2

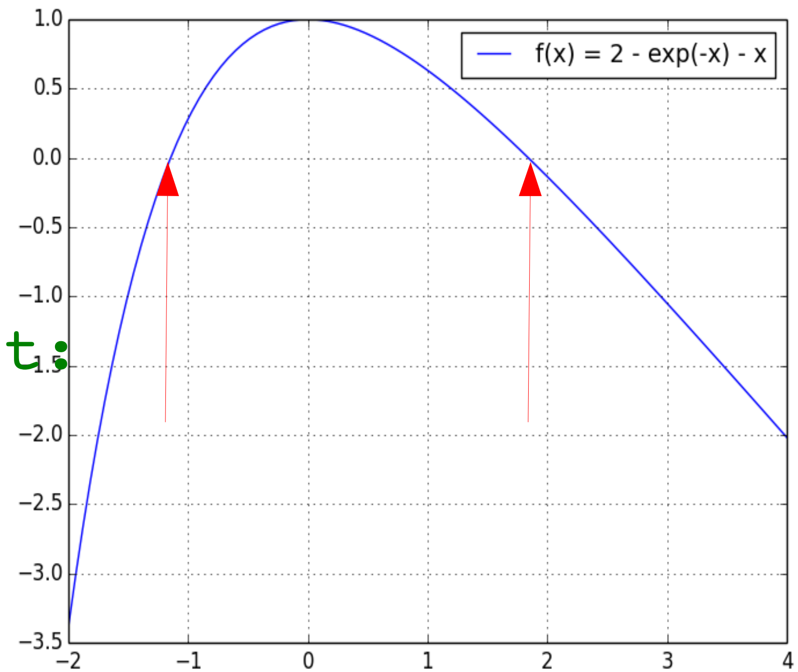


example: $f(x) = 2 - e^{-x} - x$

Bisection Method

```
target = 1e-10
xa = float( sys.argv[1] )
xb = float( sys.argv[2] )

while np.abs(xa - xb) > target:
    x = (xa + xb)/2
    if f(x)*f(xa) > 0:
        xa = x
    else:
        xb = x
print(x)
```



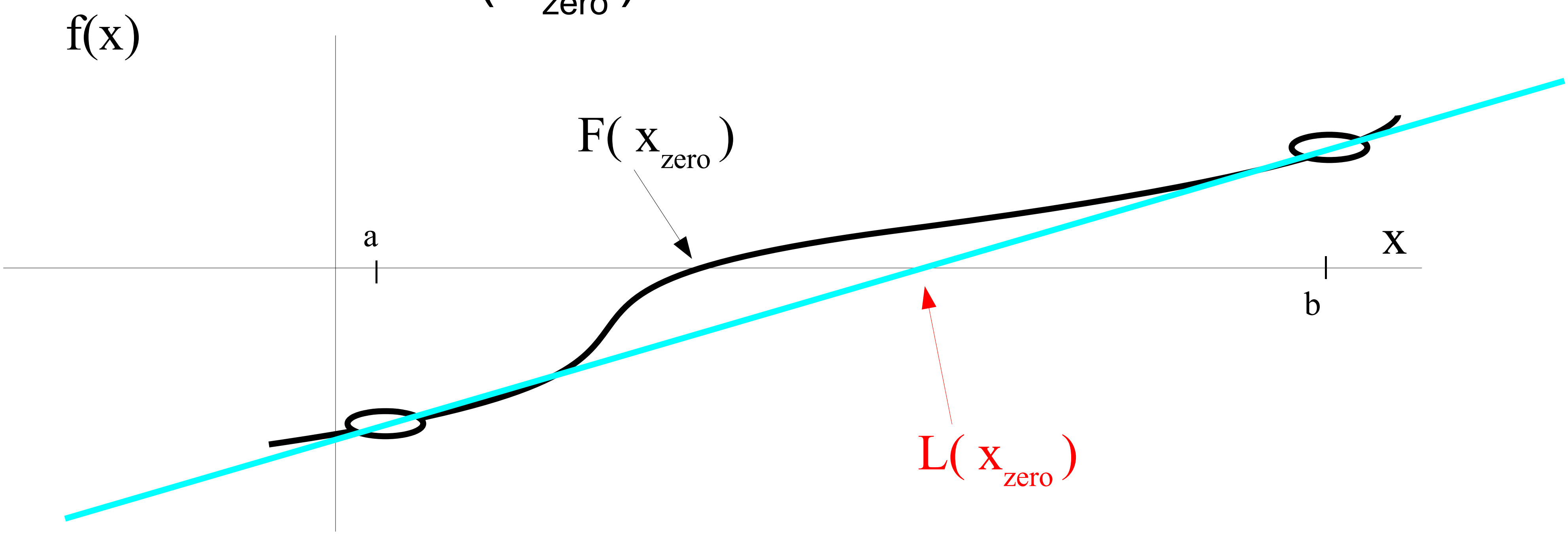
both roots are easy to find

```
hpc-login 663% bisection.py -2 -1
-1.14619322057
```

```
hpc-login 663% bisection.py  1  2
1.84140566044
```

False Position Method

- Improve rate of convergence by using information about the values of the function
- Assume the function is linear between x_a & x_b
  - use the linear zero intersection $L(x_{zero}) = 0$ to estimate $f(x_{zero}) = 0$



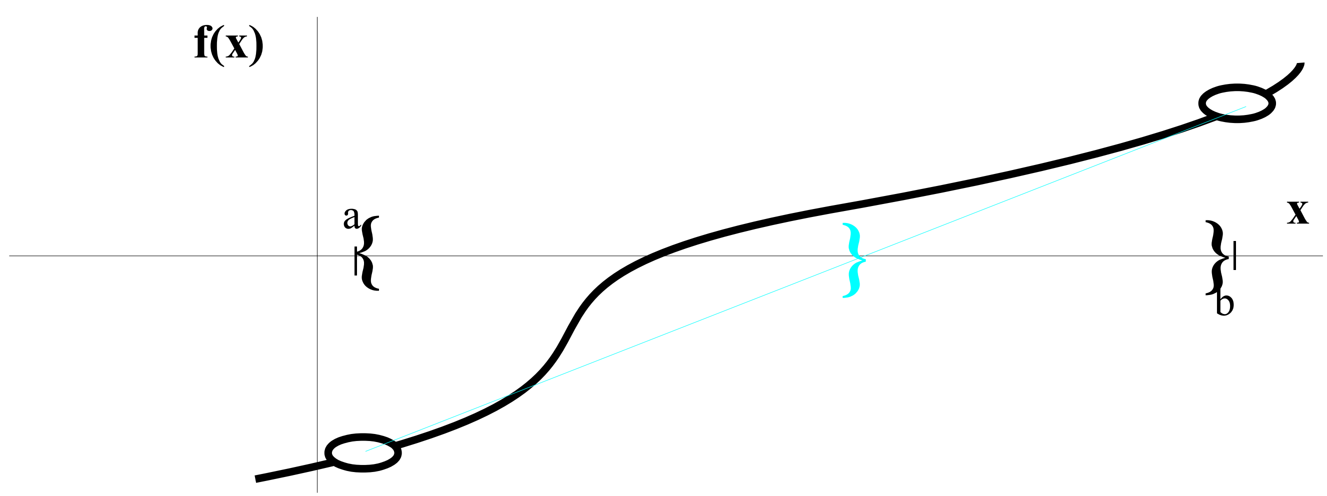
False Position Method

Basic Approach

1) Given x_a, x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the slope m and intercept b

$$m = (f(x_b) - f(x_a))/(x_b - x_a), \quad b = f(x_a) - m\, x_a$$

3) Determine linear $x_{zero} = -b/m$
   a) if $f(x_{zero}) = 0$ — stop
   b) if $f(x_{zero})*f(x_a) > 0$ — replace x_a with x_{zero}
   c) else — replace x_b with x_{zero}
4) if $|x - x_{last}| > accuracy$, repeat from step 2



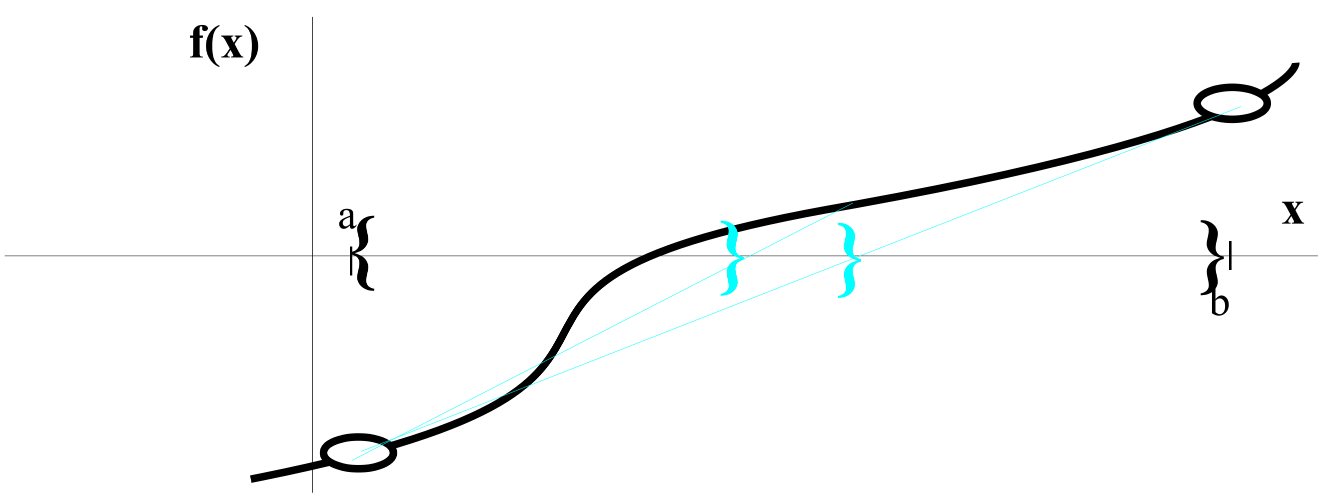
False Position Method

Basic Approach

1) Given x_a, x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the slope m and intercept b

$$m = (f(x_b) - f(x_a))/(x_b - x_a), \quad b = f(x_a) - m\, x_a$$

3) Determine linear $x_{zero} = - b/m$
   - a) if $f(x_{zero}) = 0$ stop
   - b) if $f(x_{zero})*f(x_a) > 0$ replace x_a with x_{zero}
   - c) else replace x_b with x_{zero}
4) if $|x - x_{last}| >$ accuracy, repeat from step 2



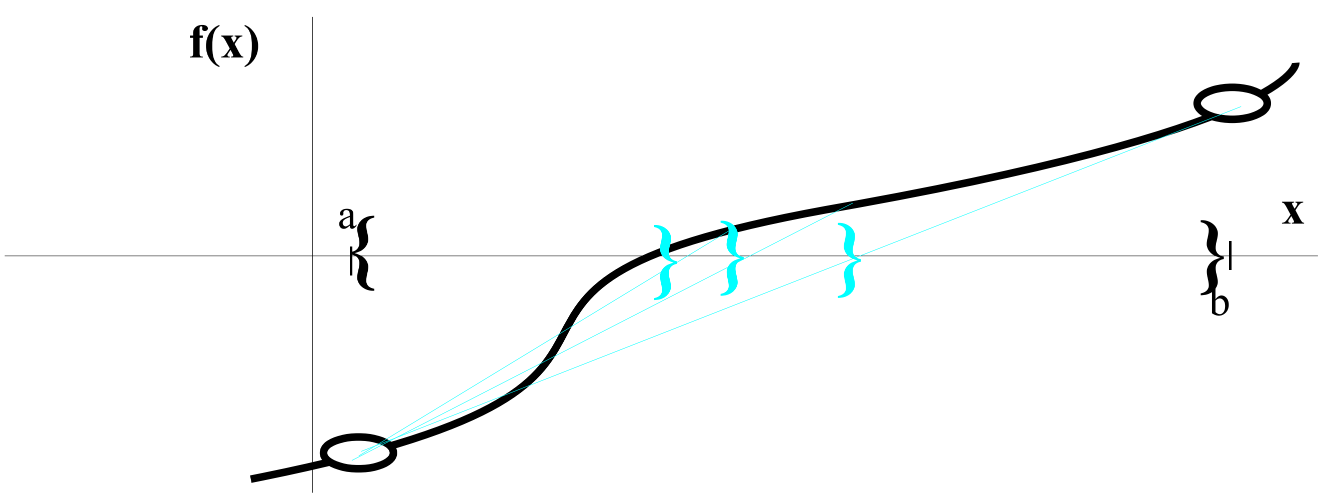
False Position Method

Basic Approach

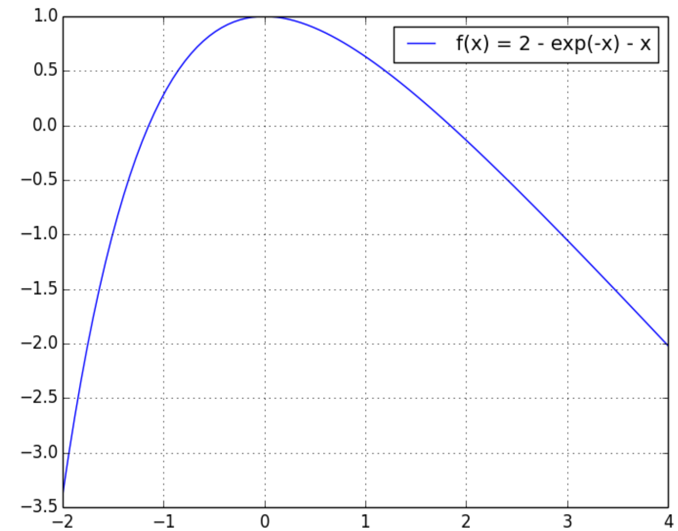
1) Given x_a, x_b check that $f(x_a)$ and $f(x_b)$ have opposite signs. Also choose a solution accuracy.
2) Calculate the slope m and intercept b

$$m = (f(x_b) - f(x_a))/(x_b - x_a), \quad b = f(x_a) - m\, x_a$$

3) Determine linear $x_{zero} = -b/m$
   a) if $f(x_{zero}) = 0$ stop
   b) if $f(x_{zero})*f(x_a) > 0$ replace x_a with x_{zero}
   c) else replace x_b with x_{zero}
4) if $|x - x_{last}| > accuracy$, repeat from step 2



example: $f(x) = 2 - e^{-x} - x$



False Position Method

```
target = 1e-10
xa = float( sys.argv[1] )
xb = float( sys.argv[2] )
xInt,xIntOld = xa,float("inf")

while np.abs(xInt - xIntOld) > target:
    xIntOld = xInt
    m = (f(xb) - f(xb)) / (xb - xa)
    yInt = f(xa) - m*xa
    xInt = -yInt/m
    if f(xInt)*f(xa) > 0:
        xa = xInt
    else:
        xb = xInt
print(xInt)
```

both roots are easy to find

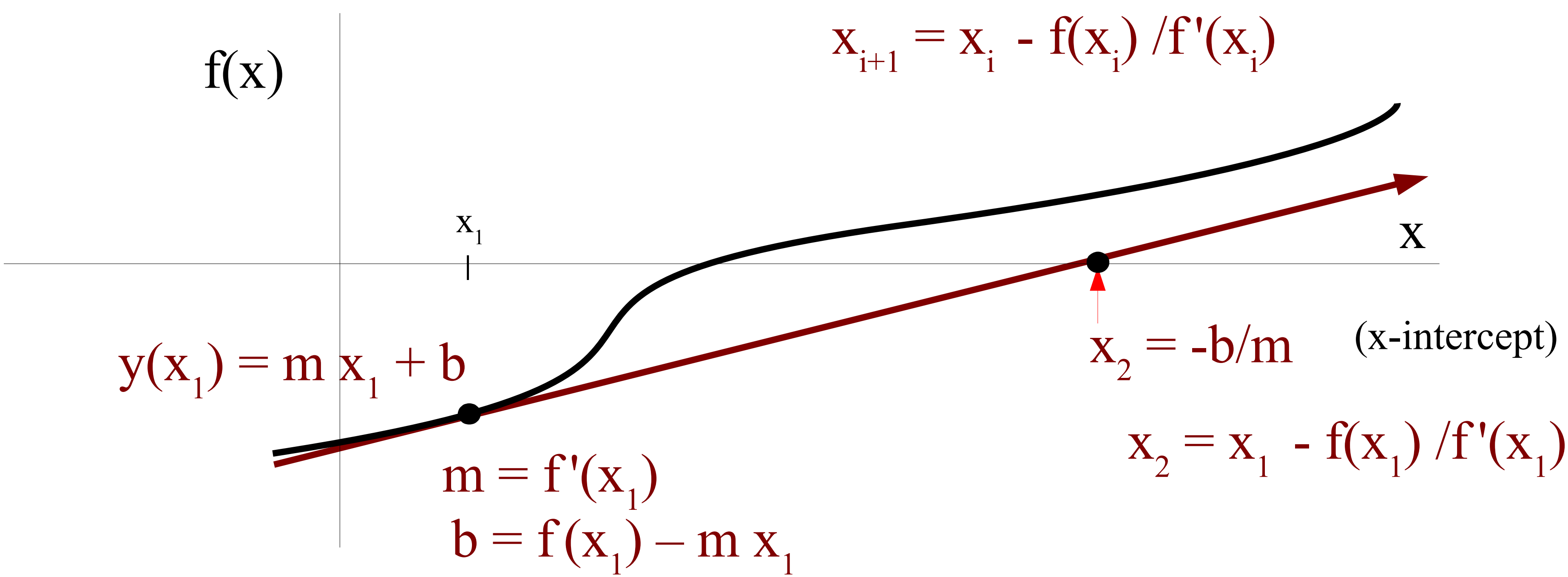
```
hpc-login 663% falseposition.py -2 -1
-1.14619322057
```

```
hpc-login 663% falseposition.py 1 2
1.84140566044
```

Newton-Raphson Method

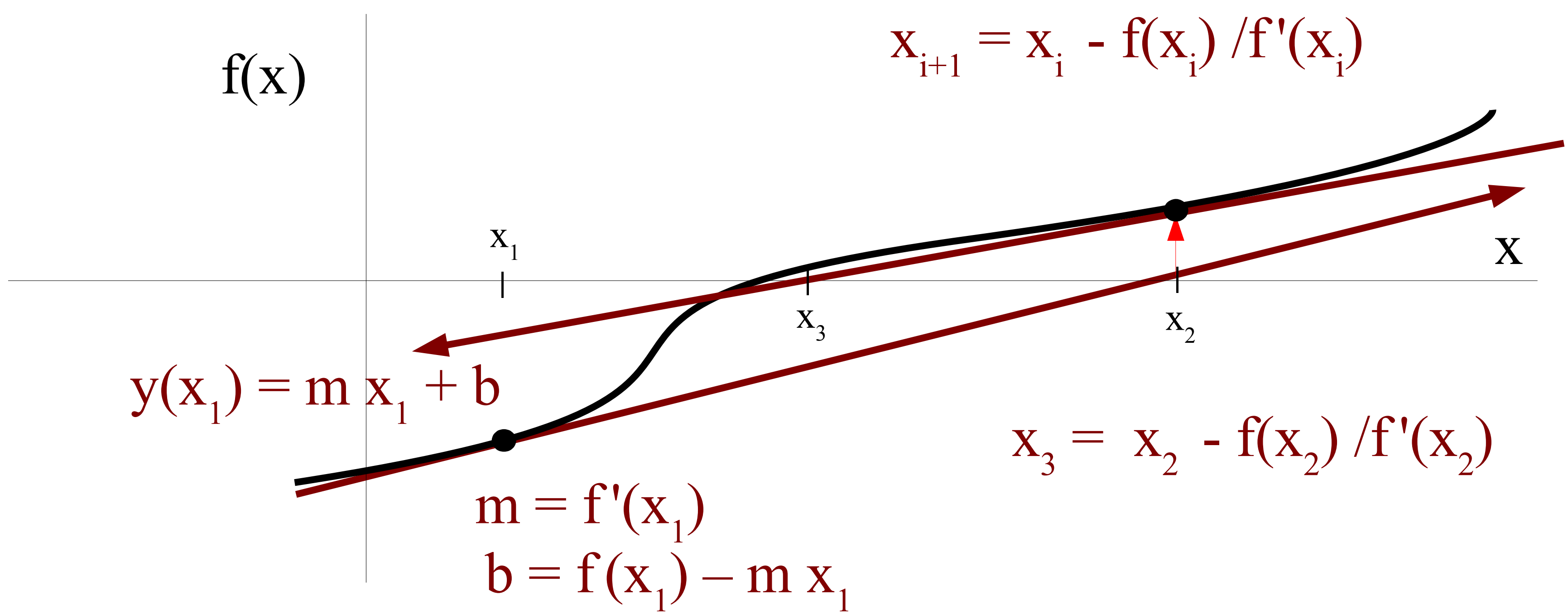
aka Newton's Method

- Most Commonly Used Root-Finding Routine
  - Uses only one starting point but needs the derivative of the function
  - Calculates $f(x_{start})$ & $f'(x_{start})$
  - Uses the tangent line's zero crossing $L_T(x_{zero})=0$ to estimate $f(x_{zero})=0$



Newton's Method

- Most Commonly Used Root-Finding Routine
  - Uses only one starting point but needs the derivative of the function
  - Calculates $f(x_{start})$ & $f'(x_{start})$
  - Uses the tangent line's zero crossing $L_T(x_{zero})=0$ to estimate $f(x_{zero})=0$



example: $f(x) = 2 - e^{-x} - x$

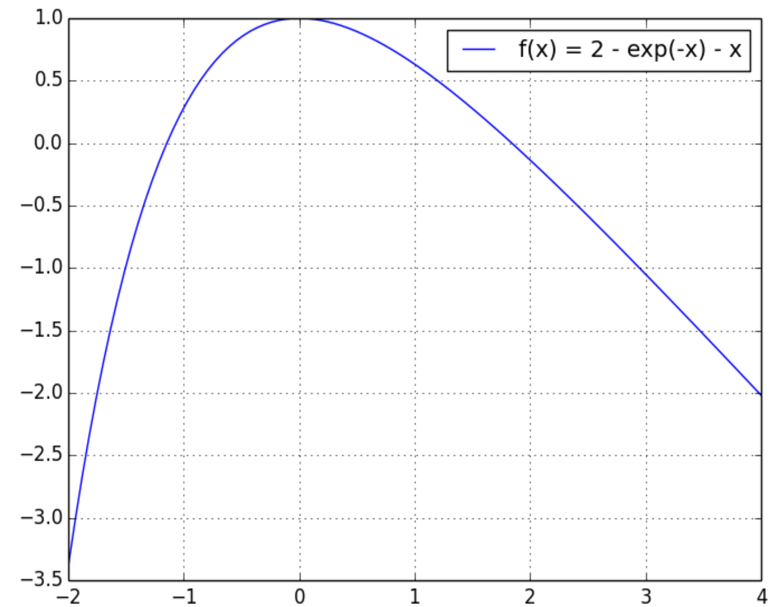
Newton's Method

```
def f(x):
    return 2 - np.exp(-x) - x

def dfdx(x):
    return np.expm1(-x)

target = 1e-10
x = float( sys.argv[1] )
xlast = float("inf")

while np.abs(x - xlast) > target:
    xlast = x
    x = xlast - f(xlast)/dfdx(xlast)
print(x)
```



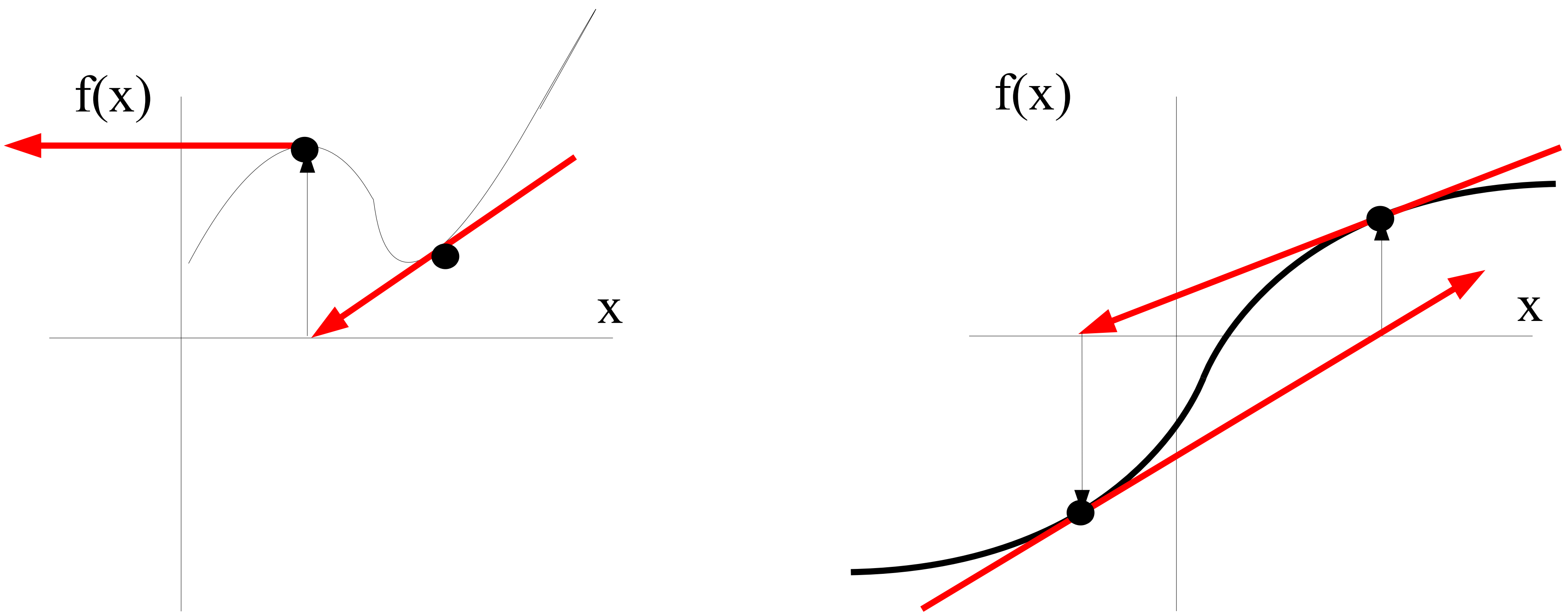
both roots are easy to find

```
# With initial x = -1.0
hpc-login 663% newton.py -1
-1.14619322062
```

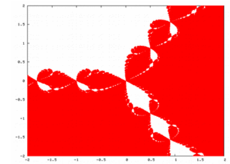
```
# With initial x = 1.0
hpc-login 663% newton.py 1
1.84140566044
```

Newton-Raphson Method

Drawbacks of the method



Newton-Raphson & Fractals



Also works for
Complex Functions (FOR FREEEE!)

$$f(z) = 0$$

$$f(z) = z^3 - 1 = 0$$

Roots:

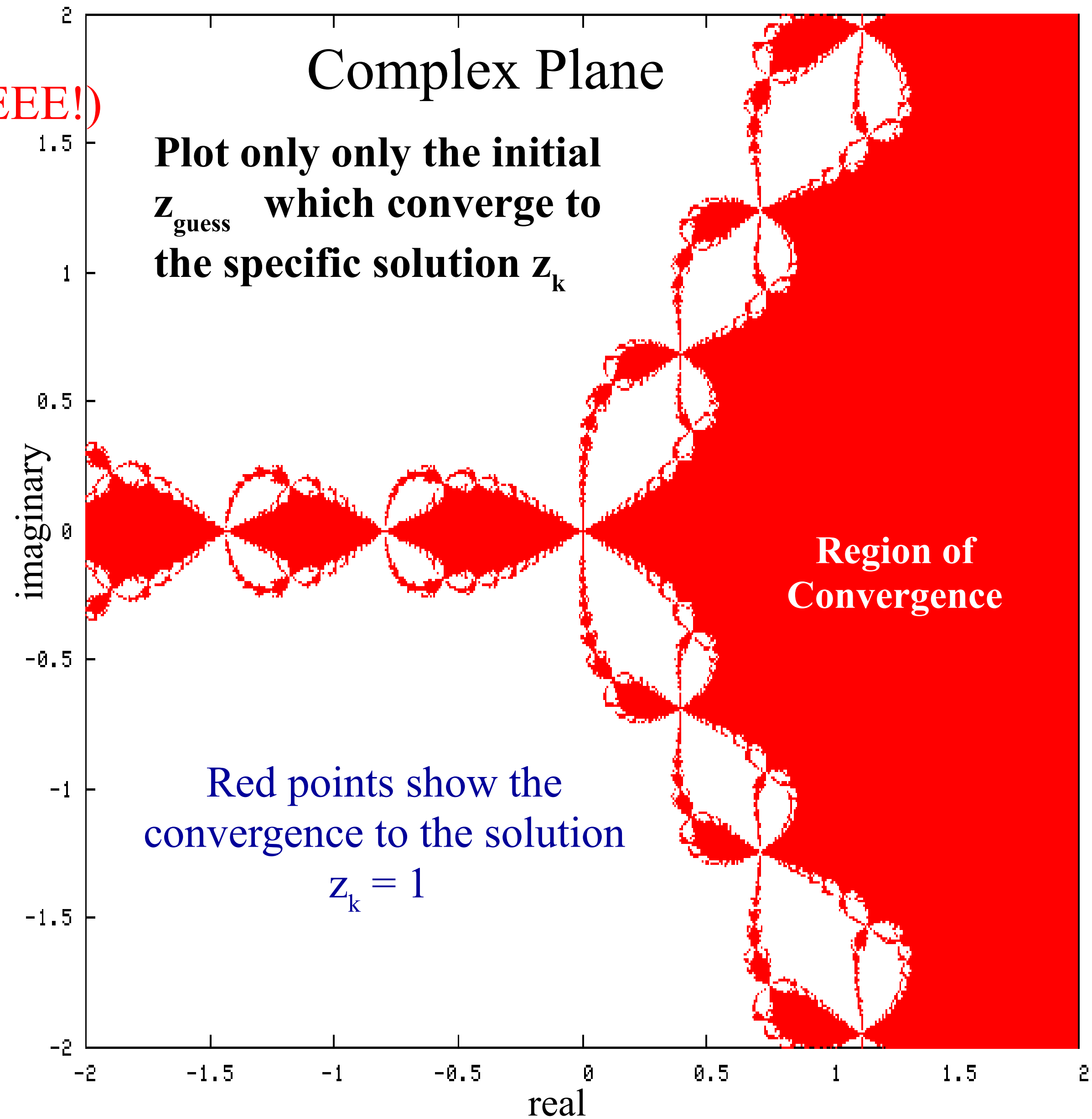
$z=1$, $z=e^{\pm 2\pi i/3}$

Newton-Raphson Method

$$z_{j+1} = z_j - (z_j^3 - 1)/(3z_j^2)$$

Look for Convergence

Convergence depends on initial "z" guess



Secant Method

- Newton's/False Position Method without a known derivative
  - Uses only two starting points, x_0 & x_1, which need not bracket the solution
  - Uses Newton's method with an approximation for f '(x)

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

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

Secant Method for guessing the root location

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

example: $f(x) = 2 - e^{-x} - x$

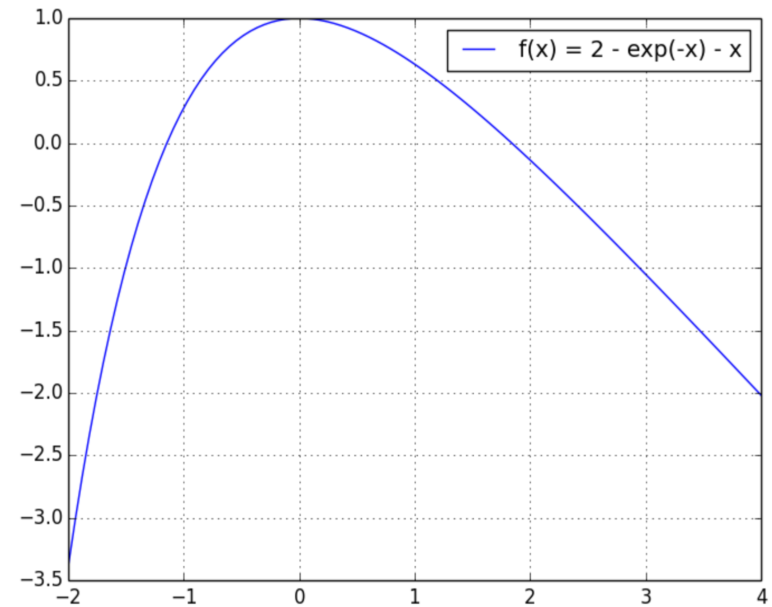
Secant Method

```
def f(x):
    return 2 - np.exp(-x) - x

def slope(y,x1,x2):
    return (y(x2)-y(x1))/(x2-x1)

target = 1e-10
xa = float( sys.arvg[1] )
xb = float( sys.arvg[2] )


while np.abs( xa-xb ) > target:
    x = xb – f(xb) / slope(f, xa, xb)
    xa, xb = xb, x
print(x)
```



both roots are easy to find

```
# With initial a,b = -0.1,-0.5

hpc-login 663% secant.py -0.1 -0.5
-1.14619322062
```

```
# With initial a,b = 0.1,0.5

hpc-login 663% secant.py 0.1 0.5
1.84140566044
```

Using Python eval() & exec()

The `eval()` allows one to execute arbitrary strings as Python code. It accepts a source string and returns an object.

```
>>> x = 1
>>> eval("x + 3")
4
>>> eval("'hello' + 'py'")
'hellopy'
>>> result = eval("2 + 4 - 3 * 3")
>>> print(result)
-3
>>> f = eval("lambda x:  x/2")
>>> print( f(11) )
5.5
```

The `exec()` allows one to execute a dynamically created statement. It accepts a source string but does not return an object.

```
>>> exec( "a = x " + "+ 10" )
>>> print(a)
11
>>>
>>>
>>> exec("def g(x):" + " 2*x")
>>> print( g(11) )
22
```

Exercise 7 Due TUESDAY March 26