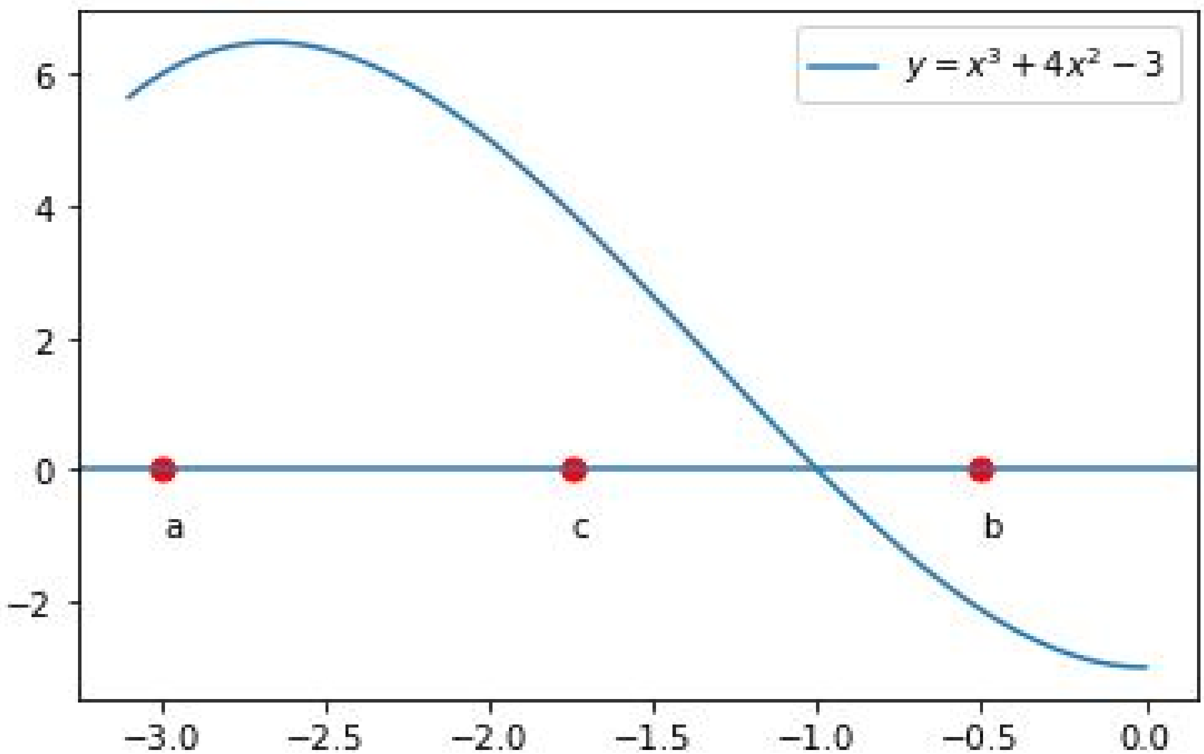
ROOT FINDING

Bisection Method

$$\mathbf{y = x^3 + 4x^2 - 3}$$

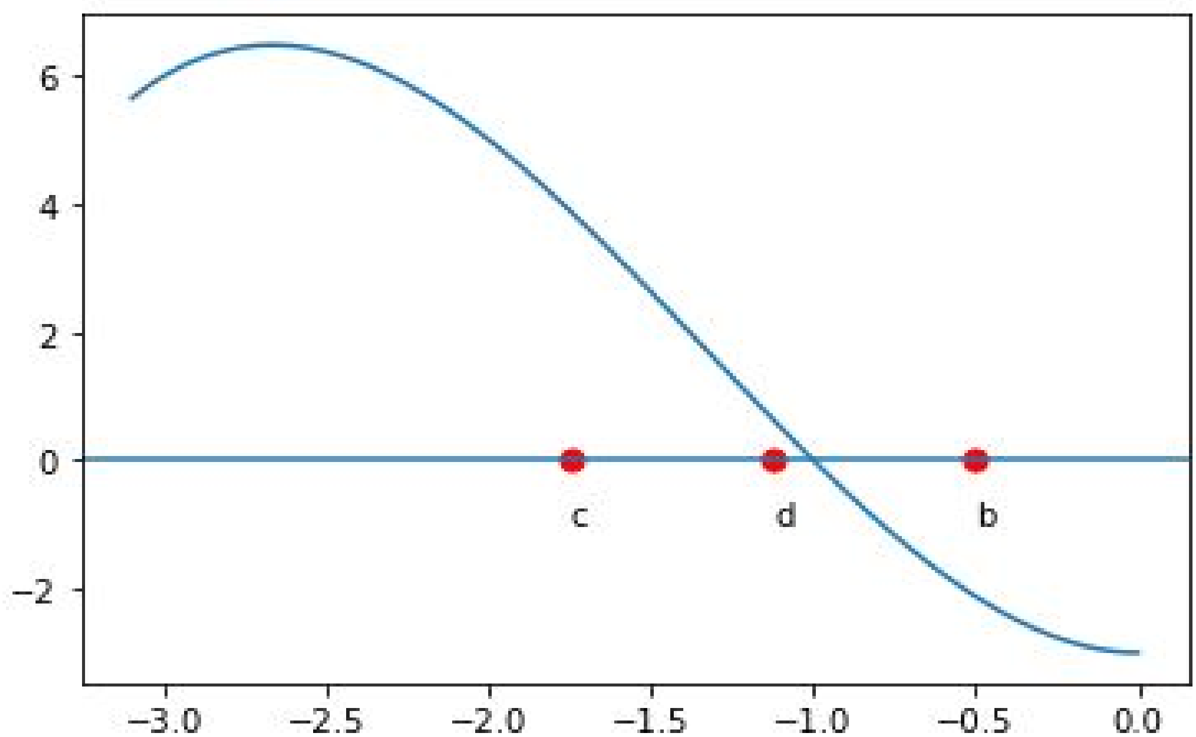


- recursively halve the intervals

https://people.duke.edu/~ccc14/sta-663-2018/notebooks/S09A_Root_Finding.html

Bisection Method (cont'd)

$$\mathbf{y = x^3 + 4x^2 - 3}$$



- slow convergence

https://people.duke.edu/~ccc14/sta-663-2018/notebooks/S09A_Root_Finding.html

Bisection Method (cont'd)

```
import matplotlib.pyplot as plt
import numpy as np
import scipy as scipy
from scipy.interpolate import interp1d

def f(x):
    return x**3 + 4*x**2 -3

x = np.linspace(-3.1, 0, 100)
plt.plot(x, x**3 + 4*x**2 -3,
        label="$y=x^{3} + 4x^{2}-3$")
plt.legend(loc="best")

a = -3.0        # initial guesses
b = -0.5        # f(a) > 0, f(b) < 0
c = 0.5*(a+b)
```

https://people.duke.edu/~ccc14/sta-663-2018/notebooks/S09A_Root_Finding.html

Bisection Method (cont'd)

```
plt.text(a,-1,"a")
plt.text(b,-1,"b")
plt.text(c,-1,"c")

plt.scatter([a,b,c],
            [f(a), f(b),f(c)],
             s=50,  facecolors='none')

plt.scatter([a,b,c], [0,0,0],
             s=50, c='red')

xaxis = plt.axhline(0)
plt.show()
```

https://people.duke.edu/~ccc14/sta-663-2018/notebooks/S09A_Root_Finding.html

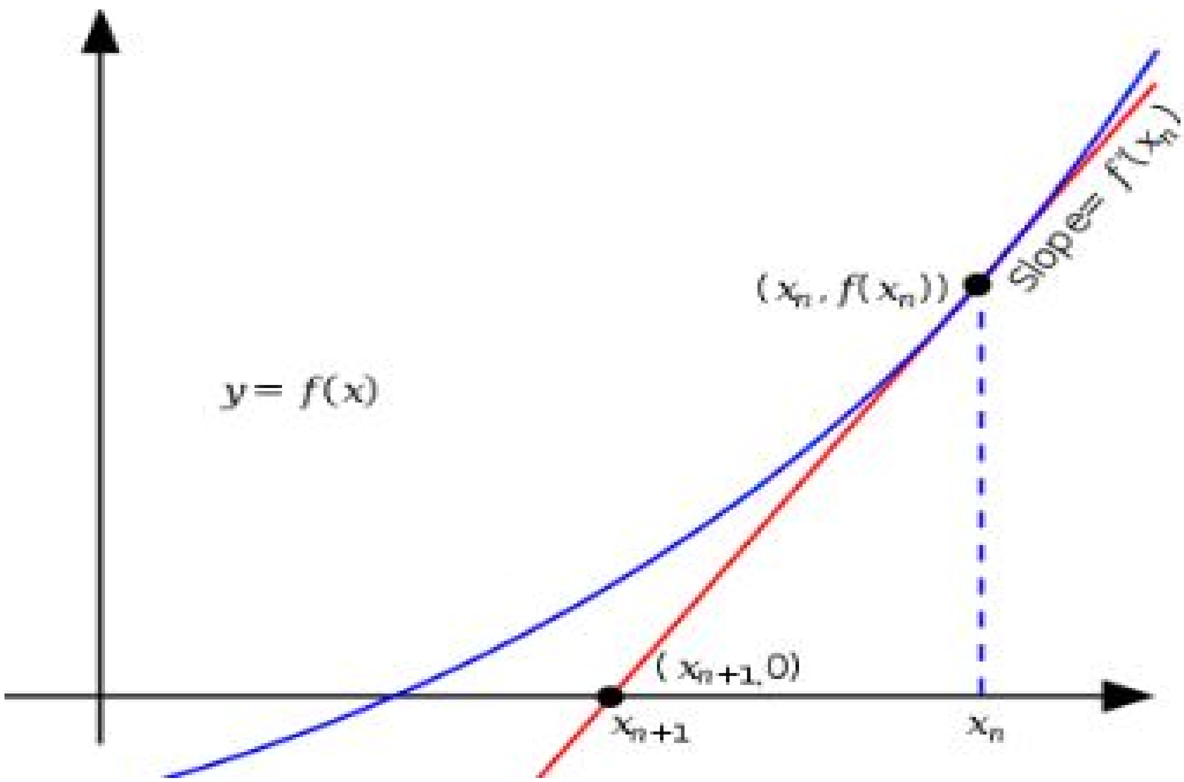
Bisection Method w. mpmath

```
import mpmath as mp
mp.dps = 2
print("using bisection method")
mp.findroot(lambda x: x**3 +4*x**2 - 3,
            [-3, 3], solver="bisect")
```

```
using bisection method
0.791287847477920003294023596864
```

Newton's Method

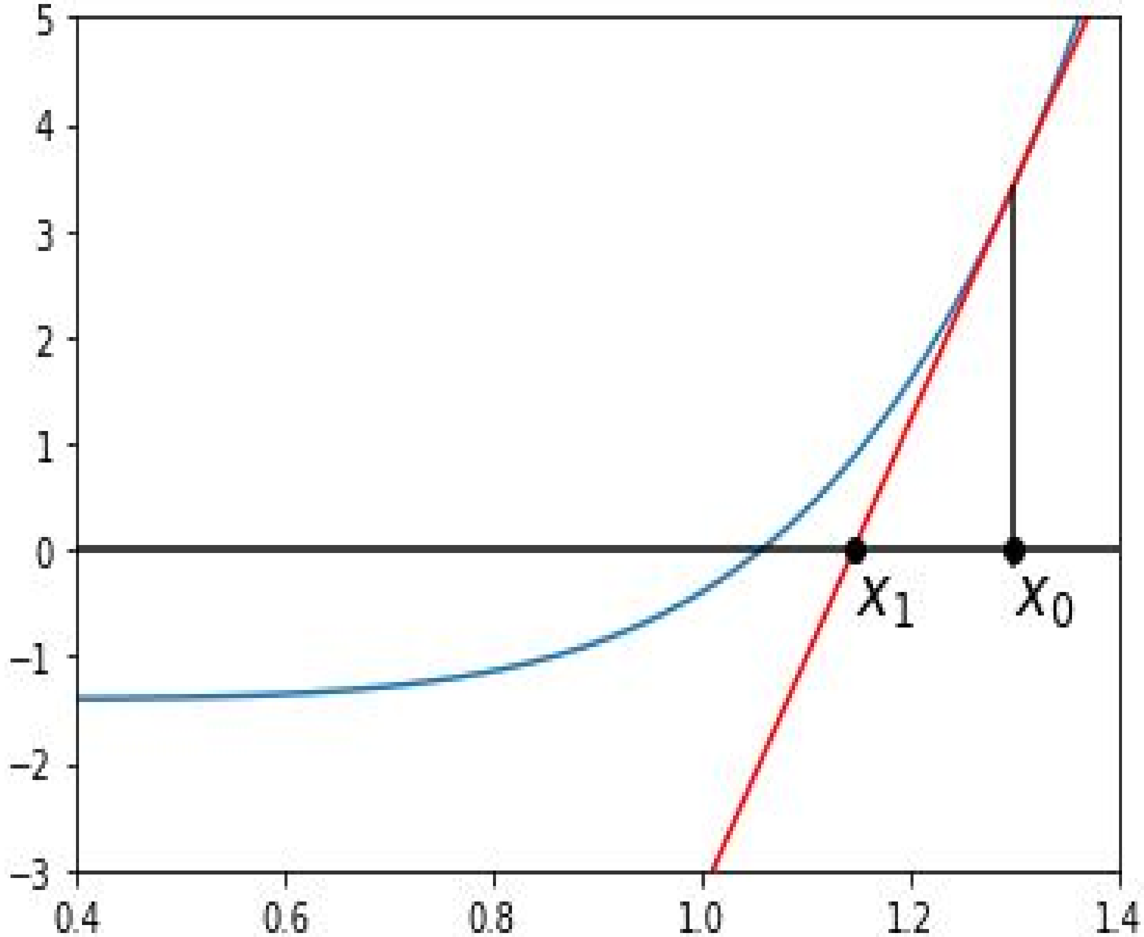
$$x_n = x_{n-1} - \frac{f\big(x_{n-1}\big)}{f'\big(x_{n-1}\big)}$$



https://en.wikipedia.org/wiki/Newton

Newton-Raphson Method

$$\mathbf{f(x) = x^6 - 1.4}$$



Newton-Raphson Method (cont'd)

```
def f(x):
    return x**6 - 1.4

def df(a):
    return (f(a+0.00001)-f(a))/0.00001

def tan_line(a):
    return df(a)*(x - a) + f(a)

def n(a):
    return a - f(a)/df(a)

x = np.linspace(-3,3,1000)
ax = plt.figure()
plt.plot(x, f(x))
plt.plot(x, tan_line(1.3),
            color = 'red')
```

Newton-Raphson Method (cont'd)

```
plt.xlim(0.4, 1.4)
plt.ylim(-3,5)
plt.axhline(color = 'black')
plt.axvline(color = 'black')
plt.axvline(1.3, 0.36, 0.8,
            color = 'black')
plt.text(1.3, -0.6, "$x_0$",
            fontsize = 18)

plt.plot(1.3, 0, 'o',
                color = 'black')
plt.plot(n(1.3), 0, 'o',
                color = 'black')
```

Newton-Raphson Method (cont'd)

```
plt.text(n(1.3), -0.6, "$x_1$",
                       fontsize = 18)
plt.title("Next Approximation",
                       loc = 'left')
plt.show()

ap = [5]
for i in range(10):
    next = ap[i] - f(ap[i])/df(ap[i])
    print("Our ", i, "approx is ",
              next)
    ap.append(next)
```

Newton-Raphson Method (cont'd)

```
0 approx is  4.166745499589058
1 approx is  3.472477858825541
2 approx is  2.894197858523393
3 approx is  2.412984746950978
4 approx is  2.0136771166978837
5 approx is  1.6851157217961754
6 approx is  1.4214393001596253
7 approx is  1.2247463131374698
8 approx is  1.1052975869999637
9 approx is  1.062525034825841
```

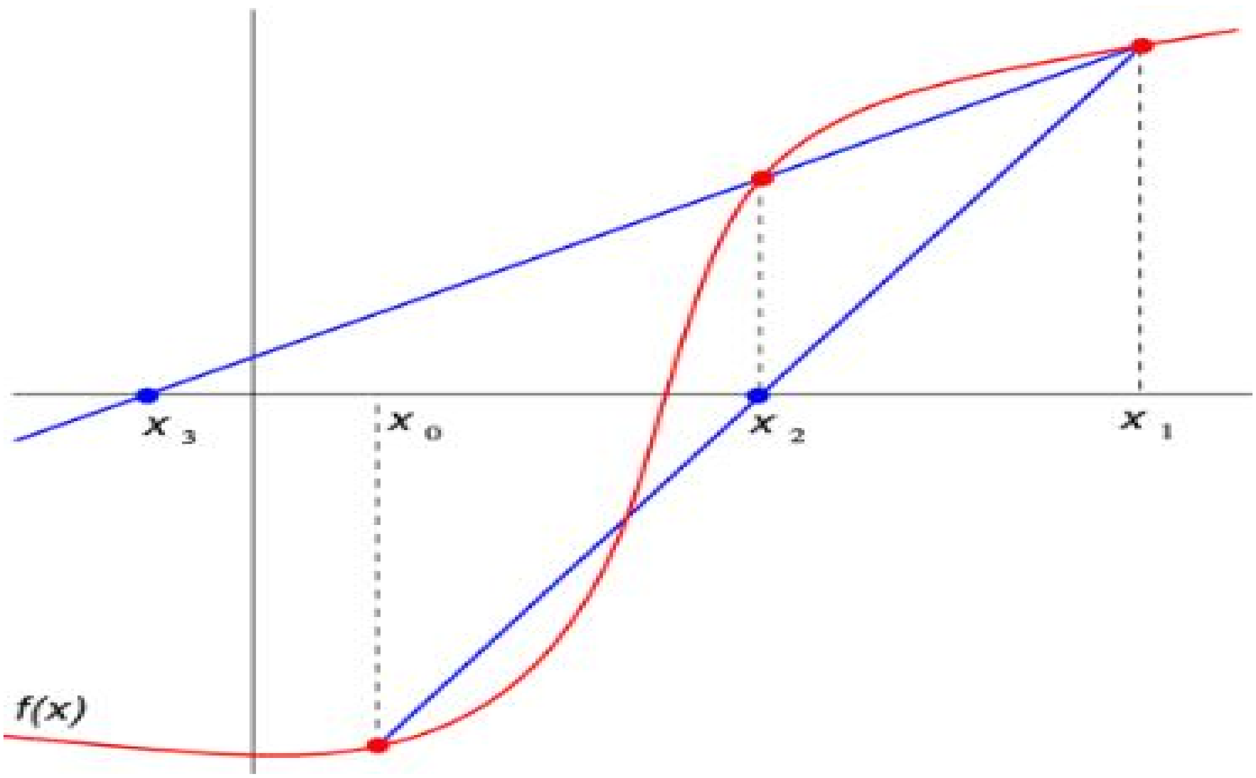
Newton's Method w. mpmath

```python
import mpmath as mp
mp.prec = 3
mp.dps = 3
print("using newton method")
mp.findroot(lambda x: x**3 +4*x**2-3,
            3, solver="anewton")
```

```
using newton method
0.791287847477920003294023596864
```

Secant Method

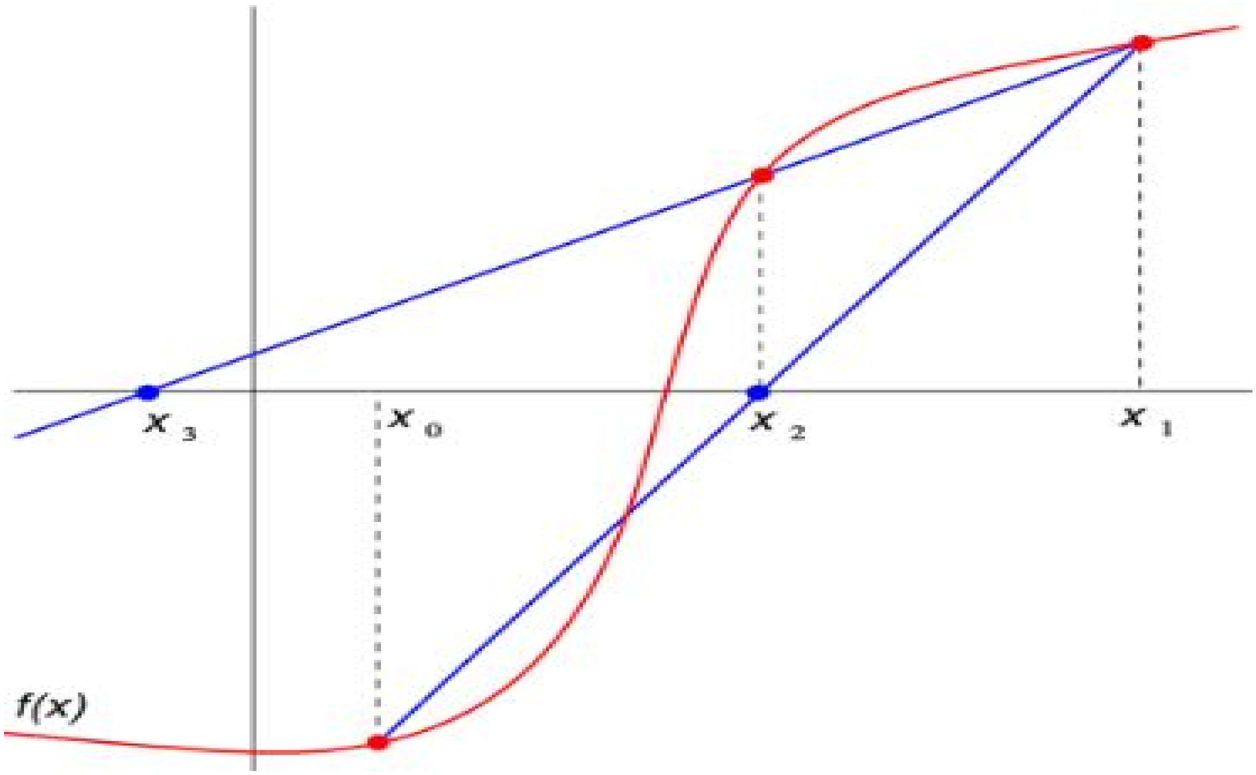
$$\mathbf{y} = \frac{\mathbf{f}(\mathbf{x_1}) - \mathbf{f}(\mathbf{x_0})}{\mathbf{x_1} - \mathbf{x_0}}(\mathbf{x} - \mathbf{x_1}) + \mathbf{f}(\mathbf{x_1})$$



https://en.wikipedia.org/wiki/Secant_method

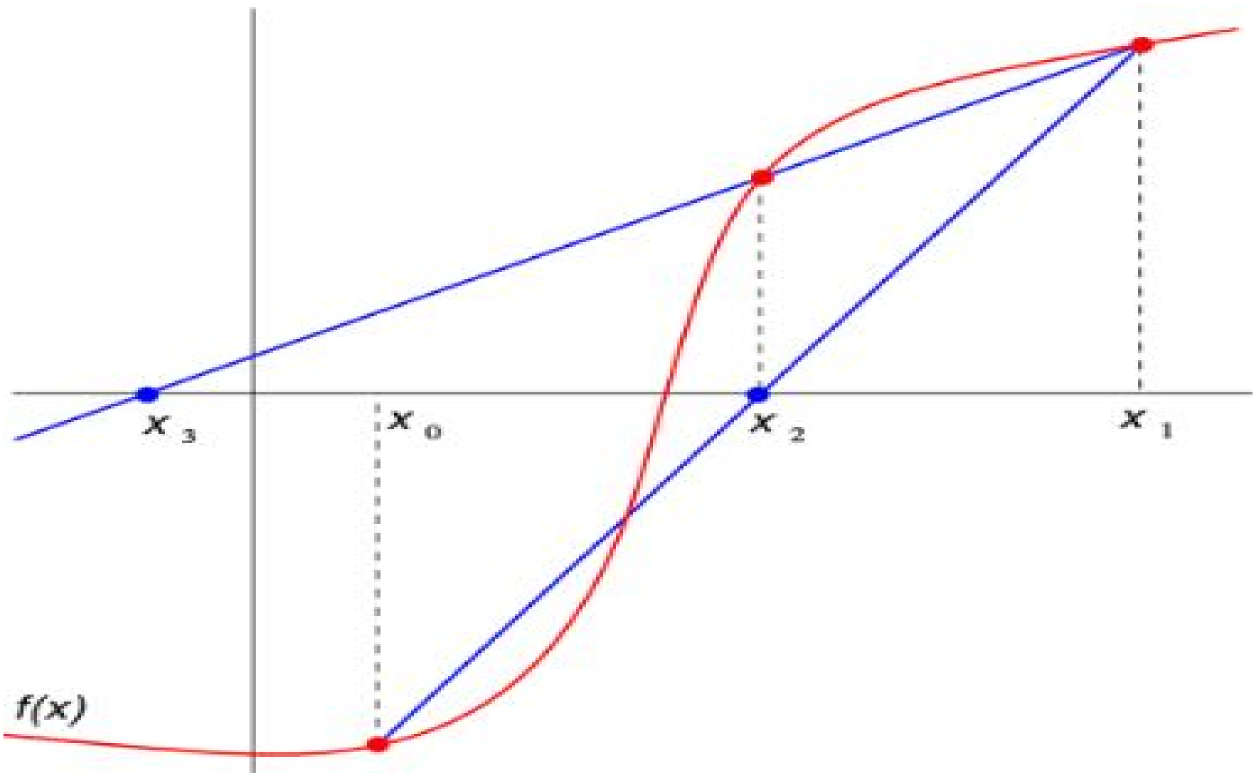
Secant Method (cont'd)

$$\mathbf{x_n = x_{n-1} - f(x_{n-1}) \cdot \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}}$$



https://en.wikipedia.org/wiki/Secant_method

Secant Method (cont'd)

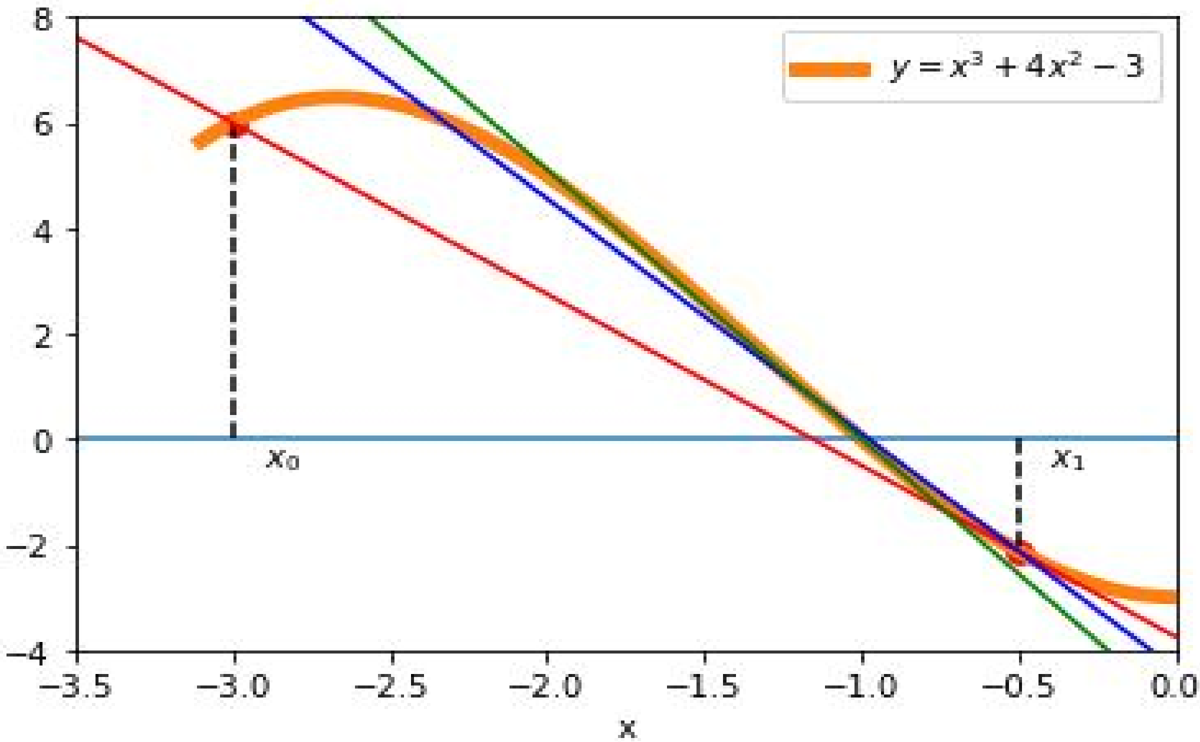


- need two initial points
- compute intersection(s)

https://en.wikipedia.org/wiki/Secant_method

Secant Method Example

$$\mathbf{y = x^3 + 4x^2 - 3}$$



- need initial points a and b

https://people.duke.edu/~ccc14/sta-663-2018/notebooks/S09A_Root_Finding.html

Secant Method Example (cont'd)

```
def f(x):
    return (x**3 + 4*x**2 -3)

x = x = np.linspace(-3.1, 0, 100)
y = f(x)

p1=plt.plot(x, y)
p1=plt.plot(x, y, lw = 5,
    label="$y=x^{3} + 4x^{2}-3$")
plt.legend(loc="best")
plt.xlim(-3.5, 0)
plt.ylim(-4, 8)
plt.xlabel('x')
plt.axhline(0)
t = np.arange(-10, 5., 0.1)

a =-3
b =-0.5
```

Secant Method Example (cont'd)

```
plt.scatter(a, f(a), s=60, color="r")
plt.scatter(b, f(b), s=60, color="r")
plt.plot([a, a], [0, f(a)], ls="--",
                 color="k")
plt.plot([b, b], [0, f(b)], ls="--",
                 color="k")
plt.text(a+0.1, -0.5,"$x_{0}$")
plt.text(b+0.1,-0.5,"$x_{1}$")

xvals = []
xvals.append(a)
xvals.append(b)
notconverge = 1
count = 0
cols=['r','b','g','y--']
```

Secant Method Example (cont'd)

```
while (notconverge==1 and count<3):
    slope=(f(xvals[count+1])-
                f(xvals[count]))/
            (xvals[count+1]-xvals[count])
    intercept=-slope*xvals[count+1]+
                    f(xvals[count+1])
    plt.plot(t, slope*t + intercept,
                         cols[count])
    nextval = -intercept/slope
    if abs(f(nextval)) < 0.001:
        notconverge=0
    else:
        xvals.append(nextval)
    count = count+1

plt.show()
```

Secant Method w. mpmath

```
import mpmath as mp
mp.prec = 3
mp.dps = 3
print("using secant method")
mp.findroot(lambda x: x**3 +4*x**2-3,
            3, solver="secant")
```

```
using newton method
0.791287847477920003294023596864
```

Rate of Convergence

$$\lim_{n \mapsto \infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^q} = \mu$$

- rate of convergence μ
- order of convergence $q \geq 1$

1. linear: $q = 1$ and $\mu < 1$
2. quadratic: $q = 2$
3. cubic: $q = 3$

- bisection: $q = 1$ (linear)
- secant: $q = (1 + \sqrt{5})/2 \approx 1.618$
- Newton-Raphson: $q = 2$ (quadratic)