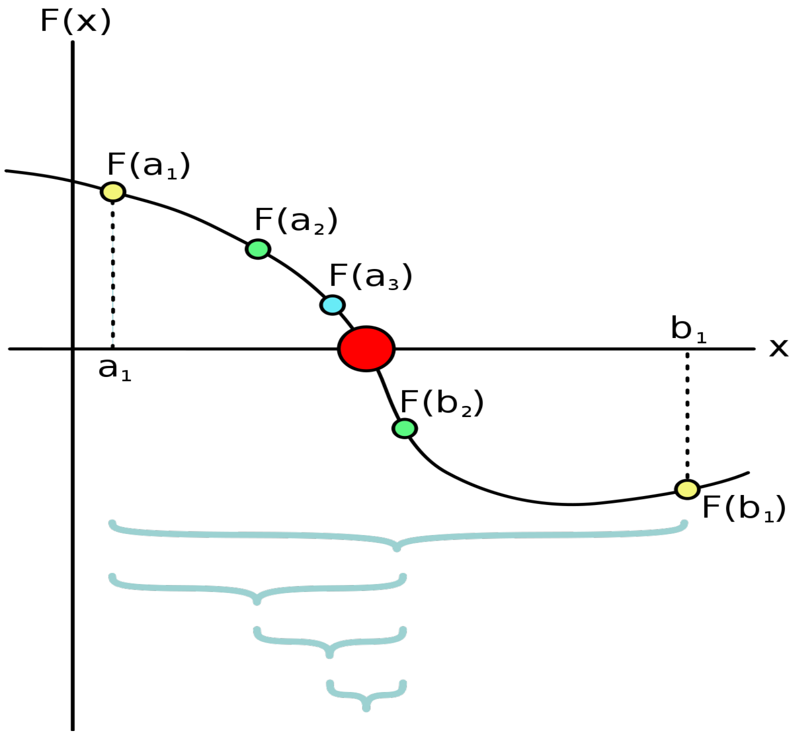
ROOT FINDING

BISECTION METHOD



Basic implementation

```
def bisection(f,a,b,N):
    for n in range(N):
        m=(a+b)/2
        if f(a)*f(m)<0:
            b=m
        elif f(m)*f(b)<0:
            a=m
    return (a+b)/2
```

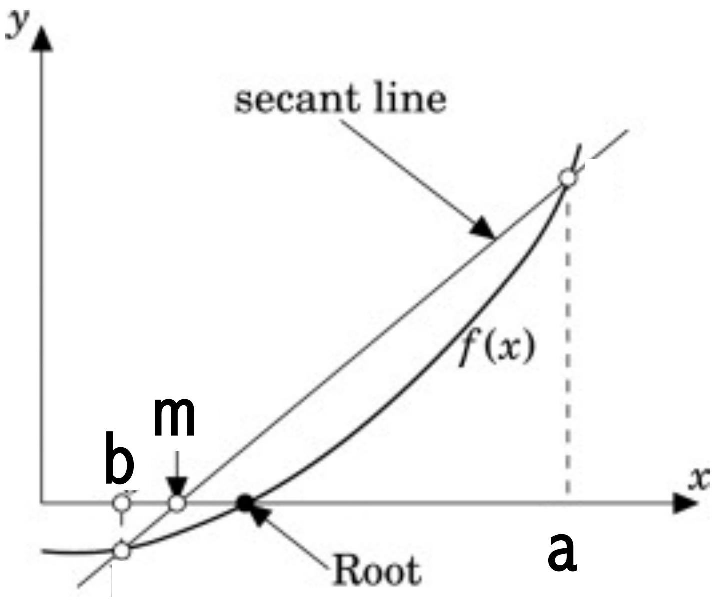
Comments

- It's a 'bracketing', iterative method
- The method is based on Bolzano's theorem
- There is a while loop implementation (stop condition)
- This method always converges (global convergence)

Algorithm

1. Start with an interval a, b such that $f(a)f(b) < 0$, which is equivalent to $f(a)$ and $f(b)$ having opposite sign. (In particular, $f(a)$ and $f(b)$ should both be not equal to zero, otherwise we are done).
2. Compute the midpoint $m = (a + b)/2$. (If $f(m)$ is equal to zero, we are done.)
3. Determine in which subinterval f changes sign:
    1. If $f(a)f(m) < 0$ then let the next interval be $[a, m]$, i.e. replace the right boundary by m by assigning $b = m$
    2. If $f(m)f(b) < 0$ then let the next interval be $[m, b]$, i.e. replace the left boundary by m by assigning $a = m$
4. Repeat steps (2.) and (3.) until the interval $[a, b]$ is sufficiently small.
5. Return the midpoint value m as an approximation of the root.

SECANT METHOD (regula falsi version)



Comments

- It's an iterative method
- This method has local convergence (if two two initial points are *close enough* to the root)

The idea

You might think that we can be smarter than choosing the midpoint. The bisection method chooses the midpoint $(a+b)/2$ no matter what values $f(a)$ and $f(b)$ take. However, if $|f(a)|$ is much smaller than $|f(b)|$ then it seems likely that the actual root is closer to a than to b. The secant method takes this into account by replacing the midpoint by the point at which the secant line connecting the endpoints $(a, f(a))$ and $(b, f(b))$ intersects with the x-axis.

The equation of the secant line passing through $(a, f(a))$ and $(b, f(b))$ is $y = f(a) + \frac{f(b)-f(a)}{b-a}(x-a)$. We want to determine the value of x such that $y = 0$, which gives $x = a - \frac{b-a}{f(b)-f(a)} f(a) = \frac{af(b)-bf(a)}{f(b)-f(a)}$. (Note that if $f(b) = -f(a)$ this reduced to the midpoint rule $x = (a+b)/2$ as it should.)

The only thing that we need to change in the bisection code is the calculation of m.

SECANT METHOD (recursive version)

The idea

- Iteratively, find the new “m” based on the old “m”
- We don’t need to consider intervals anymore

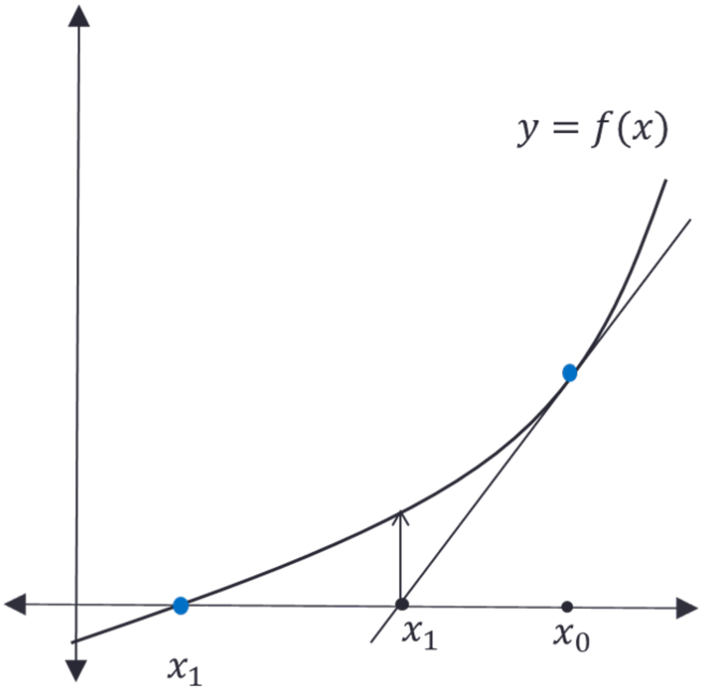
The algorithm

1. Start with an interval a, b.
2. Let $x_1 = a$ and $x_0 = b$.
3. For n from 1 to $N-1$ repeat $x_{n+1} = \frac{x_n f(x_{n-1}) - x_{n-1} f(x_n)}{f(x_{n-1}) - f(x_n)}$.
4. Return x_N

Basic implementation

```
def secant(f,a,b,N):
    x_new,x_old=a,b
    for n in range(1,N):
        x_new,x_old=(x_new*f(x_old)-x_old*f(x_new))/(f(x_old)-f(x_new)),x_new
    print("Computed approximate solution.")
    return x_new
```

NEWTON METHOD



Algorithm

1. Start with a point a.
2. Let $x_0 = a$.
3. For n from 0 to $N-1$ repeat $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
4. Return x_N

Basic implementation

```
def newton(f,df,a,N):
    x=a
    for n in range(N):
        x=x-f(x)/df(x)
    print("Computed approximate solution.")
    return x
```

Comments

- You can see it as the refinement of the secant method, where instead of a secant we consider a tangent (secant is an approximation of tangent, see numerical derivative!)
- You only need a starting point, not an interval
- Local convergence (but faster than secant)

(rough work)

SPEED OF CONVERGENCE

(that is, how fast errors shrink with number of iterations)

BISECTION METHOD $\quad |\epsilon_{n+1}| \approx C_1 |\epsilon_n|$, with $C_1 = 1/2.$ $\quad$ Linear convergence

SECANT METHOD $\quad |\epsilon_{n+1}| \approx C_2 |\epsilon_n|^{1.618}$ $\quad$ Superlinear convergence

NEWTON METHOD $\quad |\epsilon_{n+1}| \approx C_3 |\epsilon_n|^2$ $\quad$ Quadratic convergence

The main point here is that in all three cases the speed of convergence is governed by the exponent p in $|\epsilon_{n+1}| \approx C_p |\epsilon_n|^p$.

The intuitive meaning of the exponent p is that after each iteration the number of correct digits of the approximated root x_n increases roughly by a factor of p.