Root-finding

January 14, 2025

1 Root finding

Root finding algorithms aim to find the root of an equation f(x) = 0.

1.1 Bisection method

The **bisection method** is probably the simplest yet robust root-finding algorithm for solving $ f(x) = 0 $. It is a **bracketing method**—it requires that you start with an interval $[a, b]$ such that the function $ f $ has opposite signs at the endpoints (i.e., $ f(a) f(b) < 0 $). By repeatedly halving the interval and selecting the subinterval that contains the root, the method converges to a solution.

1.1.1 Steps

1. **Check the sign change**

    - Compute $ f(a) $ and $ f(b) $.

    - If $ f(a) f(b) > 0 $, then the method cannot proceed (or you need a different interval).
    - If $ f(a) = 0 $ or $ f(b) = 0 $, then you have found an exact root.

2. **Compute the midpoint**

$$m = \frac{a+b}{2}.$$

3. **Evaluate the function at the midpoint**

$$f(m) = f\left(\tfrac{a+b}{2}\right).$$

4. **Decide which subinterval contains the root**

    - If $ f(a) f(m) < 0 $, then the root lies in $[a, m]$. Set $ b ←m $.
    - Otherwise, if $ f(m) f(b) < 0 $, the root lies in $[m, b]$. Set $ a ←m $.

5. **Check for convergence**

    - If $ |b - a| < $ (your desired tolerance), or $ |f(m)| $ is sufficiently small, stop. The midpoint $ m $ is your approximate root.
    - Otherwise, check if the maximum number of iteration is within a limit (say, less than N), repeat from step 2.

[21]:
```python
from math import sin
# Example code
# Function f(x) = sin(x)
def f(x):
    return sin(x)
# to find a root between 3 and 3.25
a = 3.0
b = 3.25
eps = 1.0e-6
maxiter = 20

if abs(f(a))<eps:
    print("The root is ",a)
elif abs(f(b))<eps:
    print("The root is ",b)
elif f(a)*f(b)>0.0:
    print("Bad interval")
else:
    c = 1
    m = 0.5*(a+b)
    while abs(f(m))>eps and c<maxiter:
        print("%2d %10.6f %10.6f %10.6f %10.6f"%(c,a,b,m,f(m)))
        if f(a)*f(m)<0:
            b = m
        else:
            a = m
        c += 1
        m = 0.5*(a+b)
if c==maxiter:
    print("Did not converge")
else:
    print("The root is ",m)
```

```
 1   3.000000   3.250000   3.125000   0.016592
 2   3.125000   3.250000   3.187500  -0.045891
 3   3.125000   3.187500   3.156250  -0.014657
 4   3.125000   3.156250   3.140625   0.000968
 5   3.140625   3.156250   3.148438  -0.006845
 6   3.140625   3.148438   3.144531  -0.002939
 7   3.140625   3.144531   3.142578  -0.000985
 8   3.140625   3.142578   3.141602  -0.000009
 9   3.140625   3.141602   3.141113   0.000479
10   3.141113   3.141602   3.141357   0.000235
11   3.141357   3.141602   3.141479   0.000113
12   3.141479   3.141602   3.141541   0.000052
13   3.141541   3.141602   3.141571   0.000022
14   3.141571   3.141602   3.141586   0.000006
15   3.141586   3.141602   3.141594  -0.000001
```

```
16   3.141586   3.141594   3.141590   0.000003
The root is  3.141592025756836
```

1.2 Secant method

The **Secant method** is a root-finding algorithm for solving $ f(x) = 0 $ without needing $ f'(x) $. It uses two initial guesses, x_0 and x_1, and approximates the derivative with finite differences. The iteration step is:

$$x_{n+1} = x_n \;-\; f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

It generally converges faster than the bisection method but slower than Newton's method.

1.2.1 Steps

1. **Initial Setup**
   - Choose two initial guesses: x_0 and x_1.

   - Compute $f_0 = f(x_0)$ and $f_1 = f(x_1)$.
2. **Iteration**
   For each iteration $n = 0, 1, 2, \ldots$:
   1. **Check for division by zero**: if $|f_1 - f_0| < \varepsilon$ (some small threshold), stop.
   2. **Compute the next approximation**:

   $$x_{n+1} \;=\; x_n \;-\; f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

   3. **Evaluate** $f_{n+1} = f(x_{n+1})$.
   4. **Check convergence**: if $|x_{n+1} - x_n| <$ tolerance or if $|f(x_{n+1})| <$ tolerance, stop.
   5. **Update**: $x_{n-1} \leftarrow x_n$, $x_n \leftarrow x_{n+1}$, and similarly for f.
3. **Termination**
   - If the tolerance or maximum iterations is reached, return the last x_{n+1} as the approximate root.

[29]:
```python
from math import sin
# Example code
# Function f(x) = sin(x)
def f(x):
    return sin(x)
# to find a root between 3 and 3.25
x0 = 3.0
x1 = 3.05
eps = 1.0e-8
maxiter = 20

if abs(f(x0))<eps:
    print("The root is ",a)
elif abs(f(x1))<eps:
    print("The root is ",b)
elif abs(f(x1) - f(x0)) < eps:
    print("Flat function. May not converge.")
```

```
else:
    c = 1
    x2 = x1 - f(x1)*(x1 - x0)/(f(x1) - f(x0))
    while abs(f(x2))>eps and c<maxiter:
        print("%2d %10.6f %10.6f %10.6f %10.6f"%(c,x0,x1,x2,f(x2)))
        x0, x1 = x1, x2
        if abs(f(x1) - f(x0)) < eps:
            print("Flat function. May not converge.")
            break
        c += 1
        x2 = x1 - f(x1)*(x1 - x0)/(f(x1) - f(x0))
    if c==maxiter:
        print("Did not converge")
    else:
        print("The root is ",m)
```

```
 1   3.000000   3.050000   3.142099  -0.000507
 2   3.050000   3.142099   3.141592   0.000001
The root is  3.141592025756836
```

1.3 Newton–Raphson Method

The **Newton–Raphson method** (or **Newton's method**) is a root-finding algorithm for solving $ f(x) = 0 $ using an iterative procedure that relies on the first derivative $ f'(x) $.

1.3.1 Algorithm

1. **Initial Guess**
   Choose an initial approximation $ x_0 $ to the root.

2. **Iteration Step**
   Update $ x_{n+1} $ using:

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

3. **Check Convergence**

   - If $|x_{n+1} - x_n| <$ tolerance or $|f(x_{n+1})| <$ tolerance, stop and take $ x_{n+1} $ as the root.
   - Otherwise, continue iterating until you reach the desired accuracy or exceed the maximum iterations.

1.3.2 Notes

- **Convergence**:
  Under good conditions (i.e., if $ x_0 $ is close to the actual root and $ f'(x) 0 $ near the root), Newton's method converges *quadratically*, meaning it converges very quickly.
- **Potential Drawbacks**:
  If $ f'(x_n) $ is zero or very small, the method can fail or diverge. Good initial guesses are often crucial for success.

[28]:
```python
from math import sin, cos
# Example code
# Function f(x) = sin(x)
def f(x):
    return sin(x)
def fp(x):
    return cos(x)
# to find a root between 3 and 3.25
x0 = 3.0
eps = 1.0e-8
maxiter = 20

if abs(f(x0))<eps:
    print("The root is ",a)
elif abs(fp(x0)) < eps:
    print("Flat function. May not converge.")
else:
    c = 1
    x1 = x0 - f(x0)/fp(x0)
    while abs(f(x1))>eps and c<maxiter:
        print("%2d %10.6f %10.6f %10.6f"%(c,x0,x1,f(x1)))
        x0 = x1
        if abs(fp(x0)) < eps:
            print("Flat function. May not converge.")
            break
        c += 1
        x1 = x0 - f(x0)/fp(x0)
    if c==maxiter:
        print("Did not converge")
    else:
        print("The root is ",m)
```

```
 1   3.000000   3.142547  -0.000954
The root is  3.141592025756836
```

[ ]: