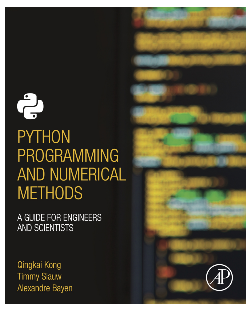


< 19.3 Bisection Method | Contents | 19.5 Root Finding in Python >

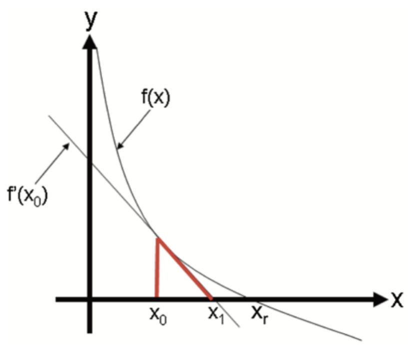
Newton-Raphson Method

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_0, 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 $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$. 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)}$.\$

An illustration of how this linear approximation improves an initial guess is shown in the following figure.



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.

TRY IT! Again, the $\sqrt{2}$ is the root of the function $f(x) = x^2 - 2$. Using $x_0 = 1.4$ as a starting point, use the previous equation to estimate $\sqrt{2}$. Compare this approximation with the value computed by Python's sqrt function.

$$x = 1.4 - \frac{1.4^2 - 2}{2(1.4)} = 1.4142857142857144$$

```python
import numpy as np

f = lambda x: x**2 - 2
f_prime = lambda x: 2*x
newton_raphson = 1.4 - (f(1.4))/(f_prime(1.4))

print("newton_raphson =", newton_raphson)
print("sqrt(2) =", np.sqrt(2))
```

```
newton_raphson = 1.4142857142857144
sqrt(2) = 1.4142135623730951
```

TRY IT! Write a function $my\_newton(f, df, x0, tol)$, where the output is an estimation of the root of *f*, *f* is a function object $f(x)$, *df* is a function object to $f'(x)$, *x0* is an initial guess, and *tol* is the error tolerance. The error measurement should be $|f(x)|$.

```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)
```

TRY IT! Use *my_newton=* to compute $\sqrt{2}$ to within tolerance of 1e-6 starting at *x0 = 1.5*.

```python
estimate = my_newton(f, f_prime, 1.5, 1e-6)
print("estimate =", estimate)
print("sqrt(2) =", np.sqrt(2))
```

```
estimate = 1.4142135623746899
sqrt(2) = 1.4142135623730951
```

If x_0 is close to x_r, then it can be proven that, in general, the Newton-Raphson method converges to x_r much faster than the bisection method. However since x_r is initially unknown, there is no way to know if the initial guess is close enough to the root to get this behavior unless some special information about the function is known *a priori* (e.g., the function has a root close to $x = 0$). In addition to this initialization problem, the Newton-Raphson method has other serious limitations. For example, if the derivative at a guess is close to 0, then the Newton step will be very large and probably lead far away from the root. Also, depending on the behavior of the function derivative between x_0 and x_r, the Newton-Raphson method may converge to a different root than x_r that may not be useful for our engineering application.

TRY IT! Compute a single Newton step to get an improved approximation of the root of the function $f(x) = x^3 + 3x^2 - 2x - 5$ and an initial guess, $x_0 = 0.29$.

```python
x0 = 0.29
x1 = x0-(x0**3+3*x0**2-2*x0-5)/(3*x0**2+6*x0-2)
print("x1 =", x1)
```

```
x1 = -688.4516883116648
```

Note that $f'(x_0) = -0.0077$ (close to 0) and the error at x_1 is approximately 324880000 (very large).

TRY IT! Consider the polynomial $f(x) = x^3 - 100x^2 - x + 100$. This polynomial has a root at $x = 1$ and $x = 100$. Use the Newton-Raphson to find a root of f starting at $x_0 = 0$.

At $x_0 = 0$, $f(x_0) = 100$, and $f'(x) = -1$. A Newton step gives $x_1 = 0 - \frac{100}{-1} = 100$, which is a root of f. However, note that this root is much farther from the initial guess than the other root at $x = 1$, and it may not be the root you wanted from an initial guess of 0.

< 19.3 Bisection Method | Contents | 19.5 Root Finding in Python >