PyNum
Interactive Studio

Numerical Analysis Series

The Secant Method

(Root Finding without Derivatives)

Author
Shahaduddin
shahaduddin.com

Date
February 2, 2026
Version 1.0

1. Theoretical Background

Introduction

Newton's method is extremely powerful but has a major weakness: the need to know the value of the derivative $f'(x)$ at each approximation. Frequently, $f'(x)$ is difficult to calculate or requires too many arithmetic operations.
To circumvent this, the **Secant Method** introduces a slight variation by approximating the derivative using a "secant line" through two previous points.

Derivation

By definition, the derivative is the limit:

$$f'(p_{n-1}) = \lim_{x \to p_{n-1}} \frac{f(x) - f(p_{n-1})}{x - p_{n-1}}$$

If p_{n-2} is close to p_{n-1}, we can approximate this as:

$$f'(p_{n-1}) \approx \frac{f(p_{n-2}) - f(p_{n-1})}{p_{n-2} - p_{n-1}} = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}$$

Substituting this approximation into Newton's formula gives the Secant formula:

$$\boxed{p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}} \tag{1}$$

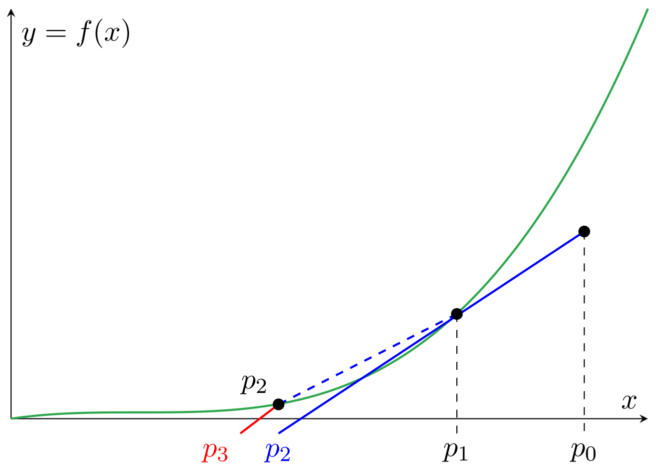

Fig 2.10. The approximation p_2 is the x-intercept of the line joining $(p_0, f(p_0))$ and $(p_1, f(p_1))$.

2. Algorithm

Secant Method Algorithm

To find a solution to $f(x) = 0$ given initial approximations p_0 and p_1:

```
1. Input: Initial approximations $p_0, p_1$; tolerance $TOL$; max iterations $N_0$.
2. Output: Approximate solution $p$ or failure message.
3. Step 1: Set $i = 2$; $q_0 = f(p_0)$; $q_1 = f(p_1)$.
4. Step 2: While $i \leq N_0$ do Steps 3–6:
     • Step 3: Set $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$.
     • Step 4: If $|p - p_1| < TOL$ then OUTPUT $p$; STOP.
     • Step 5: Set $i = i + 1$.
     • Step 6: Update variables: $p_0 = p_1, q_0 = q_1, p_1 = p, q_1 = f(p)$.
5. Step 7: Output "Method failed after $N_0$ iterations". STOP.
```

3. Code Implementations

Solving for root of $x^3 - x - 2 = 0$.

>PythonImplementation *(.py)*

```python
def secant_method(f, x0, x1, tol=1e-6, max_iter=100):
    """
    Secant Method for transcendental equations.
    Requires two initial points, no derivative needed.
    """
    for i in range(max_iter):
        fx0 = f(x0)
        fx1 = f(x1)

        if fx1 - fx0 == 0:
            raise ValueError("Division by zero")

        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)

        if abs(x2 - x1) < tol:
            return x2

        x0, x1 = x1, x2

    return x1

# Example Usage
f = lambda x: x**3 - x - 2
```

```python
root = secant_method(f, 1, 2)
print(f"Root: {root}")
```

>FortranImplementation *(.f90)*

```fortran
program secant_method
    implicit none
    real :: x0, x1, x2, tol, fx0, fx1
    integer :: i, max_iter

    ! Parameters
    x0 = 1.0; x1 = 2.0
    tol = 1e-6; max_iter = 100

    print *, "Solving x^3 - x - 2 = 0 via Secant Method..."

    do i = 1, max_iter
        fx0 = f(x0)
        fx1 = f(x1)

        if (abs(fx1 - fx0) < 1e-12) then
            print *, "Error: Vertical slope encountered."
            stop
        end if

        ! Secant Formula
        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)

        if (abs(x2 - x1) < tol) exit

        x0 = x1
        x1 = x2
    end do

    print *, "Root found: ", x2
contains
    real function f(x)
        real, intent(in) :: x
        f = x**3 - x - 2.0
    end function f
end program
```

>C++Implementation *(.cpp)*

```cpp
#include <iostream>
#include <cmath>
#include <functional>
#include <iomanip>

/**
 * Secant Method for root finding.
 * Approximates the derivative using a secant line through two points.
 */
double secant_method(std::function<double(double)> f, double x0, double x1,
    double tol = 1e-6, int max_iter = 100) {
    double x2;
    for (int i = 0; i < max_iter; ++i) {
        double fx0 = f(x0);
        double fx1 = f(x1);

        if (std::abs(fx1 - fx0) < 1e-15) {
            std::cerr << "Error: Secant slope is zero." << std::endl;
            return NAN;
        }

        // Secant Formula
        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0);

        if (std::abs(x2 - x1) < tol) return x2;

        x0 = x1;
        x1 = x2;
    }
    return x2;
}

int main() {
    auto f = [](double x) { return std::pow(x, 3) - x - 2; };
    double root = secant_method(f, 1.0, 2.0);

    std::cout << std::fixed << std::setprecision(6);
    if (!std::isnan(root)) {
        std::cout << "Root found: " << root << std::endl;
    }
    return 0;
}
```

For more numerical analysis resources, visit shahaduddin.com/PyNum