Implied Volatility Calculation

Gary Pai

September 22, 2026

1 Problem Definition

The implied volatility of an option is the volatility parameter σ that makes the theoretical option price equal to the observed market price.

For a given market option price V_{mkt}, implied volatility is defined as the solution to

$$V_{\text{model}}(\sigma) = V_{\text{mkt}}. \tag{1}$$

Equivalently, define the pricing error

$$f(\sigma) = V_{\text{model}}(\sigma) - V_{\text{mkt}}. \tag{2}$$

The implied volatility is therefore the solution of

$$f(\sigma) = 0. \tag{3}$$

Because the Black–Scholes pricing formula cannot generally be inverted analytically with respect to σ, a numerical root-finding algorithm is required.

2 Black–Scholes Option Price

For a European call option, the Black–Scholes price is

$$C = Se^{-qT}N(d_1) - Ke^{-rT}N(d_2), \tag{4}$$

where

$$d_1 = \frac{\ln(S/K) + \left(r - q + \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}}, \tag{5}$$

and

$$d_2 = d_1 - \sigma\sqrt{T}. \tag{6}$$

For a European put,

$$P = Ke^{-rT}N(-d_2) - Se^{-qT}N(-d_1). \tag{7}$$

Here,

- S is the underlying asset price,

- K is the strike price,
- r is the continuously compounded risk-free interest rate,
- q is the continuously compounded dividend yield,
- T is the time to expiration,
- σ is the volatility,
- $N(\cdot)$ is the standard normal cumulative distribution function.

The implied volatility is the value of σ satisfying

$$V_{\mathrm{BS}}(\sigma) = V_{\mathrm{mkt}}. \tag{8}$$

3 No-Arbitrage Price Bounds

Before solving for implied volatility, the market price should be checked against the theoretical no-arbitrage bounds.

For a European call,

$$\max\left(0, Se^{-qT} - Ke^{-rT}\right) \leq C \leq Se^{-qT}. \tag{9}$$

For a European put,

$$\max\left(0, Ke^{-rT} - Se^{-qT}\right) \leq P \leq Ke^{-rT}. \tag{10}$$

If the observed option price lies outside these bounds, a valid implied volatility does not exist under the Black–Scholes model.

4 Brent's Method

A robust way to solve

$$f(\sigma) = 0 \tag{11}$$

is Brent's root-finding algorithm.

Choose lower and upper volatility bounds,

$$\sigma_L < \sigma_U, \tag{12}$$

such that

$$f(\sigma_L) f(\sigma_U) < 0. \tag{13}$$

The root is then guaranteed to lie inside the interval $[\sigma_L, \sigma_U]$.

Brent's method combines three numerical techniques:

1. bisection,
2. the secant method,

3. inverse quadratic interpolation.

Bisection provides robustness by maintaining a bracket around the root, while interpolation steps can provide much faster convergence.

The iteration terminates when either

$$|f(\sigma)| < \epsilon \tag{14}$$

or

$$|\sigma_U - \sigma_L| < \epsilon, \tag{15}$$

where ϵ is the numerical tolerance.

A typical initial volatility interval is

$$\sigma_L = 10^{-4}, \qquad \sigma_U = 5. \tag{16}$$

This corresponds to a volatility range from 0.01% to 500%.

5 Newton–Raphson Method

An alternative approach is the Newton–Raphson method.

Starting from an initial guess σ_0, the iteration is

$$\sigma_{n+1} = \sigma_n - \frac{f(\sigma_n)}{f'(\sigma_n)}. \tag{17}$$

Since

$$f'(\sigma) = \frac{\partial V_{\mathrm{BS}}}{\partial \sigma}, \tag{18}$$

the derivative is the option's Vega.

For both European calls and puts,

$$\mathrm{Vega} = S e^{-qT} \phi(d_1) \sqrt{T}, \tag{19}$$

where $\phi(\cdot)$ is the standard normal probability density function.

Therefore, the Newton–Raphson iteration can be written as

$$\sigma_{n+1} = \sigma_n - \frac{V_{\mathrm{BS}}(\sigma_n) - V_{\mathrm{mkt}}}{\mathrm{Vega}(\sigma_n)}. \tag{20}$$

Newton–Raphson typically converges very quickly when the initial guess is reasonable. However, it can become unstable when Vega is very small.

For a production implied-volatility calculator, a bracketed method such as Brent's method is generally more robust.

6 Implied Volatility Algorithm

The complete implied-volatility procedure is:

1. Validate the input parameters.
2. Check the no-arbitrage bounds of the option price.
3. Define the pricing error $f(\sigma)$.
4. Choose lower and upper volatility bounds.
5. Verify that the root is bracketed.
6. Apply Brent's method.
7. Return the volatility when the convergence criterion is met.
8. Return an error if no valid solution exists.

7 JavaScript Pseudocode

The following pseudocode illustrates the overall implementation.

```
function impliedVolatility(
    marketPrice,
    S,
    K,
    r,
    q,
    T,
    optionType
) {

    // Validate input
    if (S <= 0 || K <= 0 || T <= 0) {
        return NaN;
    }

    // Check no-arbitrage bounds
    if (!isValidOptionPrice(
        marketPrice, S, K, r, q, T, optionType
    )) {
        return NaN;
    }

    // Pricing error
    function objective(sigma) {

        price = blackScholesPrice(
            S, K, r, q, T, sigma, optionType
        );

        return price - marketPrice;
```

```
    }

    // Volatility bracket
    lowerVol = 0.0001;
    upperVol = 5.0;

    fLower = objective(lowerVol);
    fUpper = objective(upperVol);

    // Root must be bracketed
    if (fLower * fUpper > 0) {
        return NaN;
    }

    // Solve f(sigma) = 0
    sigma = brentRoot(
        objective,
        lowerVol,
        upperVol,
        1e-8,
        100
    );

    return sigma;
}
```

8 Black–Scholes Pricing Function

The pricing function used by the root-finding algorithm is:

```
function blackScholesPrice(
    S, K, r, q, T, sigma, optionType
) {

    d1 =
        (ln(S / K)
        + (r - q + 0.5 * sigma * sigma) * T)
        / (sigma * sqrt(T));

    d2 = d1 - sigma * sqrt(T);

    if (optionType === "call") {

        return S * exp(-q * T) * N(d1)
             - K * exp(-r * T) * N(d2);

    } else {

        return K * exp(-r * T) * N(-d2)
             - S * exp(-q * T) * N(-d1);
    }
}
```

9 Brent Root-Finding Pseudocode

A simplified version of the root-finding procedure is:

```
function brentRoot(
    f,
    lower,
    upper,
    tolerance,
    maxIterations
) {

    a = lower;
    b = upper;

    fa = f(a);
    fb = f(b);

    if (fa * fb > 0) {
        return NaN;
    }

    for (iteration = 1;
         iteration <= maxIterations;
         iteration++) {

        // Choose interpolation or bisection
        if (interpolationIsReliable) {
            x = interpolationStep(a, b, fa, fb);
        } else {
            x = 0.5 * (a + b);
        }

        fx = f(x);

        // Convergence test
        if (abs(fx) < tolerance ||
            abs(b - a) < tolerance) {

            return x;
        }

        // Keep the root bracketed
        if (fa * fx < 0) {

            b = x;
            fb = fx;

        } else {

            a = x;
            fa = fx;
        }
    }
```

```
    return NaN;
}
```

The pseudocode above illustrates the main idea of Brent's method. A production implementation should use the complete Brent algorithm, including its interpolation safeguards.

10 Numerical Considerations

For the standard Black–Scholes model, the option price is monotonically increasing with volatility for positive S, K, and T.

Thus,

$$\frac{\partial C}{\partial \sigma} > 0 \tag{21}$$

and

$$\frac{\partial P}{\partial \sigma} > 0. \tag{22}$$

Consequently, when the market price lies strictly inside the no-arbitrage bounds, there is generally a unique positive implied volatility.

Numerical precision becomes particularly important when the option is very close to expiration or is extremely deep in or out of the money. The implementation should also avoid evaluating the Black–Scholes formula directly at $\sigma = 0$.

11 Summary

Implied volatility is obtained by numerically inverting the option pricing model:

$$\boxed{V_{\mathrm{BS}}(\sigma) = V_{\mathrm{mkt}}} \tag{23}$$

or equivalently,

$$\boxed{f(\sigma) = V_{\mathrm{BS}}(\sigma) - V_{\mathrm{mkt}} = 0.} \tag{24}$$

Brent's method provides a robust solution by maintaining a bracket around the root while using interpolation to accelerate convergence. Newton–Raphson can be faster but requires Vega and can become unstable when Vega is small.