Numerical Methods in Black-Scholes Model

Ching-Yi Wang

March 2024

1 Introduction

The Black-Scholes equation is a cornerstone in the field of financial mathematics, providing a theoretical framework for pricing European call and put options. A critical component of this model is the implied volatility, which represents market expectations of the underlying asset's volatility. However, the Black-Scholes equation does not offer an explicit solution for implied volatility, necessitating the use of numerical methods to solve for this parameter. Traders and risk analysts use this implied volatility to assess the market's expectations and make informed decisions about option trading and risk management. Root finding methods help in determining the implied volatility efficiently, allowing financial professionals to adjust their strategies, hedge risks, and optimize their portfolios in response to market dynamics. The accurate calculation of implied volatility is crucial for pricing options accurately and managing the associated financial risks effectively. The main objective of this project is to implement and analyze four numerical methods - the Bisection method, Newton's method, Fixed Point iteration, and the Secant method - to find the implied volatility in the Black-Scholes model. The project aims to evaluate these methods based on their convergence speed, accuracy, and computational efficiency, providing insights into their applicability in financial modeling and risk management.

Equation:

$$C(S, K, T, r, \sigma) - P = 0$$

Where:

C is the call option price. P is the put option price. S is the current stock price. K is the option strike price. T is the time to maturity. r is the risk-free interest rate. is the implied volatility.

Let's consider some numerical values for the variables other than in the Black-Scholes equation to calculate the implied volatility (σ).

Please note that these values are for illustration purposes, and actual market scenarios would involve real-time data.

Let's assume the following parameter values:

Call option price (P): 8

Current stock price (S): 100

Option strike price (K): 95

Time to maturity (T): 0.5 years

Risk-free interest rate (r): 0.02

Implied volatility (σ) :

To be determined.

Now, using the Black-Scholes equation:

C(S,K,T,r,)P=0

Black-Scholes (100 , 95 , 0.5 , 0.02 , σ) - 8 = 0

Black-Scholes(100,95,0.5,0.02,σ) - 8=0

The Black-Scholes formula for European call options is expressed as follows:

$C(S, K, T, r, \sigma) = S \cdot N(d_1) - K \cdot e^{-rT} \cdot N(d_2)$

Here's a breakdown of each component:

- $(C(S, K, T, r, \sigma))$: Call option price.
- (S): Current stock price.
- (K): Option strike price.
- (T): Time to maturity (in years).
- (r): Risk-free interest rate.
- (σ): Implied volatility.

The terms (d_1) and (d_2) are calculated using the cumulative distribution function (CDF) of the standard normal distribution (often denoted as $(N(\cdot))$.

$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$

$d_2 = d_1 - \sigma\sqrt{T}$

In these formulas:

- (ln): Natural logarithm. - (e): Euler's number (approximately 2.71828).

In these formulas:

(ln) : Natural logarithm. - (e): Euler's number (approximately 2.71828). The Black-Scholes formula provides the theoretical price of a European call option based on the specified parameters.

2 Matlab Code for Secant Method

Function (Secant):

```
1
2  function p = Secant(p0, p1, call_price, S, K, T, r, tol, Nmax)
3      n = 0;
4      p = [p0, p1];
5
6      while n < Nmax
7          p(end + 1) = p(end) - ((black_scholes_call(S, K, T, r, p(end)) -
       call_price) * (p(end) - p(end - 1))) / ...
8              ((black_scholes_call(S, K, T, r, p(end)) - call_price) - (
       black_scholes_call(S, K, T, r, p(end - 1)) - call_price));
9
10         if abs(p(end) - p(end - 1)) < tol
11             return;
12         end
13
14         n = n + 1;
15     end
16
17     error('Function does not converge within the maximum iteration limit.');
18 end
```

Main Script:

```
1  % Set MATLAB display precision
2  format long;
3
4  % Given parameters
5  call_option_price = 8.0;
6  current_stock_price = 100.0;
7  strike_price = 95.0;
8  time_to_maturity = 0.5;
9  risk_free_interest_rate = 0.02;
10
11 % Find the root using the secant method
12 p_values = Secant(0.0001, 0.5, call_option_price, current_stock_price,
       strike_price, time_to_maturity, risk_free_interest_rate, 0.000001, 1000);
13
14 % Calculate fcn values
15 fcn_values = arrayfun(@(sigma) black_scholes_call(current_stock_price,
       strike_price, time_to_maturity, risk_free_interest_rate, sigma), p_values)
       ;
16
17 % Calculate differences
18 diff_values = [p_values(1), diff(p_values)];
19
20 % Create a table
21 secant_table = table((0:numel(p_values)-1)', p_values', fcn_values',
       diff_values', 'VariableNames', {'iteration', 'p_values', 'fcn_values', '
       diff_values'});
22
23 % Display the table
24 disp(secant_table);
```

Secant Method Output

Command Window

```
>> SM
    iteration        p_values           fcn_values          diff_values
    _________    ________________    ________________    ________________

        0                0.0001    5.94526579382902                0.0001
        1                   0.5    16.7890296437313                0.4999
        2        0.0948237180634585    6.58136946891915      -0.405176281936541
        3         0.151133923750295    7.76846388240308      0.0563102056868361
        4         0.162116914031202    8.02583297346784      0.0109829902809077
        5         0.161014515544459    7.99975881928655     -0.00110239848674321
        6         0.161024712510071    7.99999976728866     1.01969656114143e-05
        7         0.161024722358459    8.00000000000212     9.84838863238302e-09
```

2.1 Discussion for Secant Method

Pros: Efficiency and Simplicity The secant method on the Black-Scholes model in our specific case converges in 6 iterations, which is fairly fast. The secant method has advantages other than its convergence speed, such as its simplicity. It does not require the calculation of derivative like Newton's method, nor does it converge slowly like the bisection method. It is a well-balanced algorithm that features both speed and simplicity.

Cons: Initial guess issues Firstly, the secant method requires two distinct initial guesses. For any target value of the root that cannot be intuitively approximated, initial guesses that are too far apart from each other or from the actual root can result in slower convergence or divergence. Another problem with the initial guess is division by zero. As can be seen in the specific example iteration I ran, the first initial guess is 0.0001, due to the fact that I found out that 0 cannot be a part of the calculation.

3 Newton's Method

Newton's Method Formula:

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

Variables:
- x_n is the current guess for the root.
- x_{n+1} is the next guess.
- $f(x)$ is the function whose root we are trying to find.
- $f'(x)$ is the derivative of $f(x)$, with respect to x.

3.1 Matlab Code for Newton's Method

Function (NM_call1):

```
function call = NM_call1(S, K, T, r, sigma)
    d1 = (log(S / K) + (r + sigma^2 / 2) * T) / (sigma * sqrt(T));
    d2 = d1 - sigma * sqrt(T);
    call = S * normcdf(d1) - K * exp(-r * T) * normcdf(d2);
end
```

Function (NM_call2):

```

function v = NM_call2(S, K, T, r, sigma)
    d1 = (log(S / K) + (r + sigma^2 / 2) * T) / (sigma * sqrt(T));
    v = S * normpdf(d1) * sqrt(T);
end
```

Function (Newton Volatility)

```
    function [vol, data] = newton_volatility(C, S, K, T, r)
    guess = 1;  % Initial guess for volatility
    tolerance = 1e-6;  % Tolerance for stopping the search
    i = 0;
    data = [];
    while true
        i = i + 1;
        price = NM_call1(S, K, T, r, guess);
        v = NM_call2(S, K, T, r, guess);

        % Avoid division by zero
        if v == 0
            break;
        end
        error = (price - C) / v;
        % Log iteration details
        data(i,:) = [guess, price, error];
        % Check for convergence
        if abs(error) < tolerance
            break;
        end
        guess = guess - error;  % Update guess
    end
    vol = guess;
end
```

Parameter's value (Black Scholes Equation):

$$\text{call\_option\_price} = 8.0 = \text{P}$$
$$\text{current\_stock\_price} = 100.0 = \text{S}$$
$$\text{strike\_price} = 95.0 = \text{K}$$
$$\text{time\_to\_maturity} = 0.5 = \text{T}$$
$$\text{risk\_free\_interest\_rate} = 0.02 = \text{r}$$

Main Script:

```
% Example values (adjust these with real market data)
C = 8;  % Call option price
S = 100;  % Current stock price
K = 95;  % Strike price of the option
T = 0.5;    % Time to maturity in years
r = 0.02;  % Risk-free interest rate

% Find the implied volatility
[vol, data] = newton_volatility(C, S, K, T, r);

% Display the results
disp('Volatility: ');
disp(vol);
disp('Iteration Data: ');
disp(data);
```

Newton's Method Output

Command Window

```
>> NM
Volatility:
   0.161024786361928

Iteration Data:
   1.000000000000000  29.880355618298061   0.854562773531167
   0.145437226468833   7.637278378299925  -0.015856178675810
   0.161293405144643   8.006350381500468   0.000268618782715
   0.161024786361928   8.000001512378390   0.000000064003554
```

3.2 Discussion for Newton's Method

Result report: Our interval for the σ (implied volatility) is [0, 2] To utilize Newton's method, we decided to set the initial guess at $\sigma = 1$ which is the midpoint of the interval, to optimize our chances of converging to the correct solution efficiently. The next step is to test the convergence we need to test whether the function is differentiable on $\sigma = 1$ and the result shows that the derivative of the function is 25.604152551468523 which means that the function is continuous. Therefore, we can do further iterations to shrink the interval. After 3 more iterations we got $\sigma = 0.161024786361928$ which has a relatively small error of less than 10^{-8}. Therefore, Newton's Method has successfully found the root and demonstrated good efficiency in finding roots for non-linear equations, particularly in the context of financial models like the Black-Scholes formula.

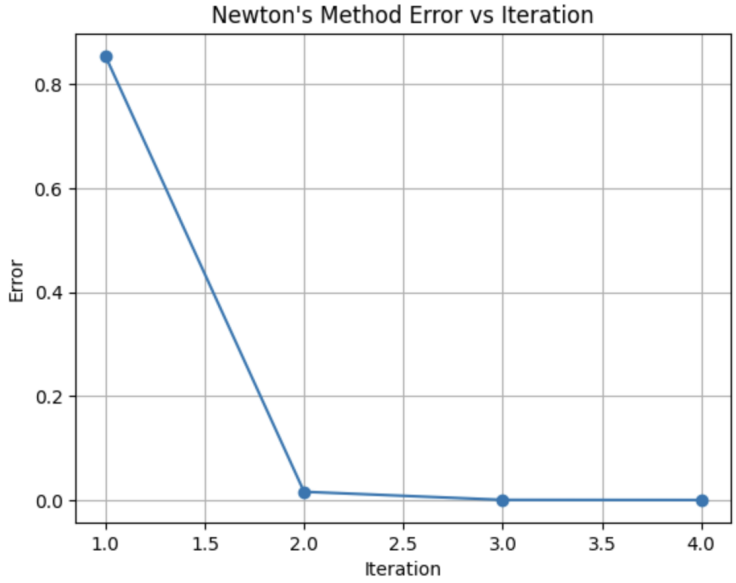
Pros: Simplicity, Efficiency, High Precision

This model applies Newton's Method to find the σ (implied volatility). Newton's Method is generally efficient with less computation compared to other root-finding methods (Bisection, FPI, Secant) which provide a fast root-finding time, especially when the initial guess is close to the actual root.

In addition, Newton's Method provides a very high precision with relatively less iteration, this could decrease the error due to the computation such as rounding error and calculation error by reduce the computationally expensive.

Cons: Initial Guess Sensitivity

However, there might be an extreme case where the initial guess is far from the actual implied volatility, and the function becomes in-differentiable where $f'(x) = 0$ which can not be converged due to the discontinuous result.



4 Bisection Method

The Bisection method is a numerical technique used to find the root of a function within a given interval. It is particularly useful for finding the root of a continuous function where the root lies within a known interval, and the function changes sign over that interval.

Here's how the Bisection method works:

1. **Initialization**: Begin with an interval $[a, b]$ where the function changes sign (i.e., $f(a)$ and $f(b)$ have opposite signs), ensuring that the root lies within this interval.
2. **Iteration**: Divide the interval in half and evaluate the function at the midpoint $c = \frac{a+b}{2}$.
3. **Update Interval**: Determine in which half of the interval the root lies based on the sign of $f(c)$. If $f(c)$ has the same sign as $f(a)$, update the interval to $[c, b]$; otherwise, update it to $[a, c]$.
4. **Repeat**: Repeat steps 2 and 3 until the interval becomes sufficiently small or the desired level of accuracy is achieved.
5. **Output**: The final value of the midpoint c is an approximation of the root of the function within the specified tolerance.

4.1 Matlab Code for Bisection Method

Function (black_scholes_call):

```
function call_price = black_scholes_call(S, K, T, r, sigma)
    d1 = (log(S / K) + (r + 0.5 * sigma^2) * T) / (sigma * sqrt(T));
    d2 = d1 - sigma * sqrt(T);
    call_price = S * normcdf(d1) - K * exp(-r * T) * normcdf(d2);
end
```

Function (final_implied_volatility):

```
function final_implied_volatility = implied_volatility_bisection_call(
    call_price, S, K, T, r, a, b, tol, max_iter)
    fprintf('Iteration\tan\t\tbn\t\tcn\t\tabs(fcn)\n');

    fa = call_price - black_scholes_call(S, K, T, r, a);
    fb = call_price - black_scholes_call(S, K, T, r, b);

    if sign(fa) == sign(fb)
        error('Initial bounds [a, b] must enclose the root.');
    end

    for n = 1:max_iter
        c = (a + b) / 2;
        fc = call_price - black_scholes_call(S, K, T, r, c);

        fprintf('%d\t\t%.9f\t%.9f\t%.9f\t%.9f\n', n, a, b, c, abs(fc));

        if abs(fc) < tol
            final_implied_volatility = c;
            return;
        end

        if sign(fc) == sign(fa)
            a = c;
            fa = fc;
        else
            b = c;
            fb = fc;
        end
    end

    error('The bisection method did not converge.');
end
```

Main Script:

```
% Set MATLAB display precision
format long;

% Main script
% Given parameters
call_option_price = 8.0;
current_stock_price = 100.0;
strike_price = 95.0;
time_to_maturity = 0.5;
risk_free_interest_rate = 0.02;

% Find implied volatility
final_implied_volatility = implied_volatility_bisection_call(
    call_option_price, current_stock_price, strike_price, time_to_maturity,
    risk_free_interest_rate, 0.01, 2.0, 1e-6, 100);

% Display the final implied volatility
fprintf('Final Calculated Implied Volatility: %.9f\n',
    final_implied_volatility);
```

Bisection Method Output

Command Window

Iteration	an	bn	cn	abs(fcn)
1	0.010000000	2.000000000	1.005000000	22.008338708
2	0.010000000	1.005000000	0.507500000	8.988018435
3	0.010000000	0.507500000	0.258750000	2.436924834
4	0.010000000	0.258750000	0.134375000	0.612138831
5	0.134375000	0.258750000	0.196562500	0.861778788
6	0.134375000	0.196562500	0.165468750	0.105415509
7	0.134375000	0.165468750	0.149921875	0.259589301
8	0.149921875	0.165468750	0.157695312	0.078435050
9	0.157695312	0.165468750	0.161582031	0.013175494
10	0.157695312	0.161582031	0.159638672	0.032711102
11	0.159638672	0.161582031	0.160610352	0.009787797
12	0.160610352	0.161582031	0.161096191	0.001688892
13	0.160610352	0.161096191	0.160853271	0.004050697
14	0.160853271	0.161096191	0.160974731	0.001181213
15	0.160974731	0.161096191	0.161035461	0.000253762
16	0.160974731	0.161035461	0.161005096	0.000463745
17	0.161005096	0.161035461	0.161020279	0.000104996
18	0.161020279	0.161035461	0.161027870	0.000074382
19	0.161020279	0.161027870	0.161024075	0.000015307
20	0.161024075	0.161027870	0.161025972	0.000029537
21	0.161024075	0.161025972	0.161025023	0.000007115
22	0.161024075	0.161025023	0.161024549	0.000004096
23	0.161024549	0.161025023	0.161024786	0.000001509
24	0.161024549	0.161024786	0.161024668	0.000001293
25	0.161024668	0.161024786	0.161024727	0.000000108

Final Calculated Implied Volatility: 0.161024727

4.2 Discussion of Results

Our function is:

$$f(\sigma) = S \cdot N\left(\frac{\ln\left(\frac{K}{S}\right) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right) - K \cdot e^{-rT} \cdot N\left(\frac{\ln\left(\frac{K}{S}\right) + (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right) - P$$

In the code above, we set $a_1 = 0.01$ and $b_1 = 2.0$. We calculated the values of $f(a_1)$ and $f(b_1)$, given the provided parameters:

$$\begin{gathered}\text{call\_option\_price} = 8.0 = \text{P} \\ \text{current\_stock\_price} = 100.0 = \text{S} \\ \text{strike\_price} = 95.0 = \text{K} \\ \text{time\_to\_maturity} = 0.5 = \text{T} \\ \text{risk\_free\_interest\_rate} = 0.02 = \text{r}\end{gathered}$$

The results are:

$$f(a_1) = -22.008338708$$

$$f(b_1) = 1.200062517$$

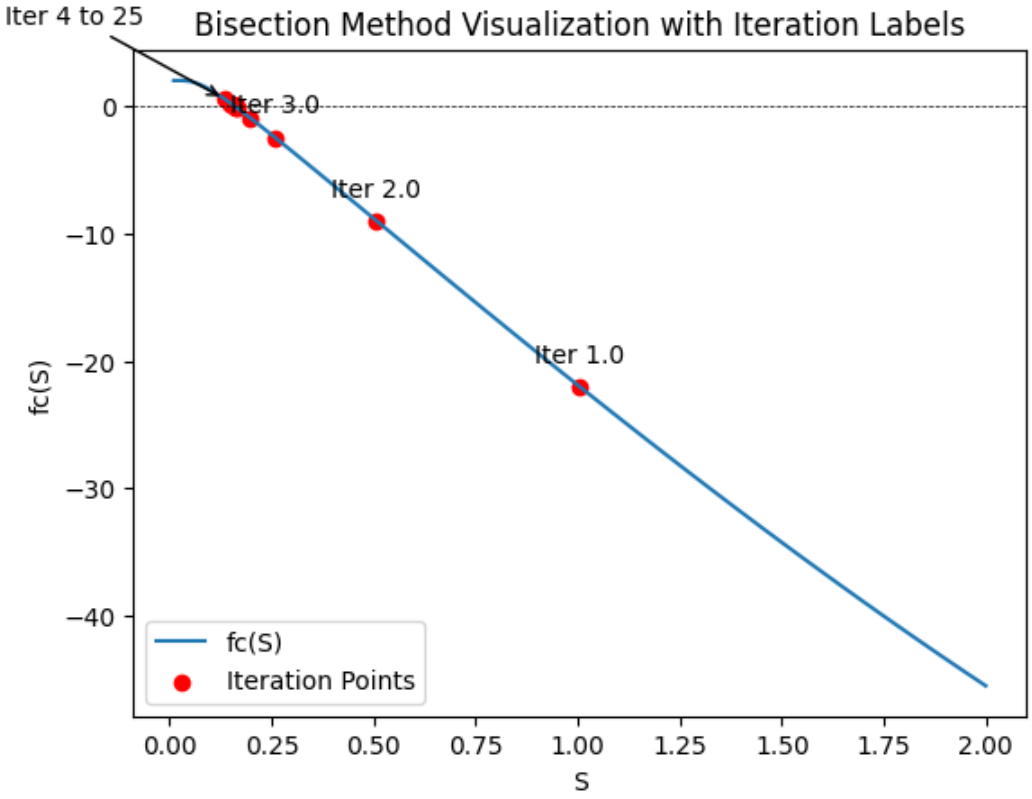
Since $f(a_1) \times f(b_1) = (-22.008338708) \times (1.200062517) < 0$, it confirms that $f(a_1)$ and $f(b_1)$ have opposite signs, indicating that a root exists within the interval $[0.01, 2.0]$.

Now, let's discuss the results of the provided code, which uses the Bisection method to find the implied volatility of a call option using the Black-Scholes formula:

- **Bisection Method**: The `implied_volatility_bisection_call` function uses the Bisection method to find the implied volatility of the call option by iteratively adjusting the volatility until the theoretical option price matches the observed option price.
- **Results**: The code iterates through the Bisection method and prints a table showing the iteration number, interval bounds (a_n, b_n), midpoint c_n, and absolute value of the function evaluated at the midpoint $|f(c_n)|$. Finally, it prints the final calculated implied volatility.
- **Discussion**: The Bisection method iterates through 25 iterations to converge to a final implied volatility of approximately 0.16102473. Each iteration refines the interval where the root (implied volatility) lies until absolute value of the function evaluated at the midpoint $|f(c_n)|$ is below the specified tolerance level (10^{-6}). The Bisection method demonstrates its effectiveness in finding the implied volatility, providing a numerical solution to an otherwise complex problem in options pricing.
- **Pros of Bisection Method**:
  - Guaranteed Convergence: The Bisection method guarantees convergence to a solution as long as the function is continuous and changes sign over the interval.
  - Simplicity: The method is relatively simple to implement and understand, making it accessible to a wide range of users.

- Robustness: Bisection is robust and stable, making it less prone to numerical errors compared to some other iterative methods.

- **Cons of Bisection Method**:
  - Slow Convergence: Bisection may converge slowly, especially for functions with complex behavior or narrow convergence intervals.
  - Limited Applicability: The method is most suitable for finding a single root within a given interval and may not be efficient for functions with multiple roots or other types of problems.
  - Requires Interval Knowledge: Bisection requires initial knowledge of an interval containing the root, which may not always be readily available or easy to determine.



5 Fixed-point iteration Method

Methodology

Fixed-Point Iteration Formula:

$$x_{n+1} = g(x_n)$$

Variables:

- x_n is the current guess for the root.
- x_{n+1} is the next guess.
- $g(x)$ is a function reformulated from the original function $f(x)$, where $f(x) = 0$, such that $x = g(x)$.

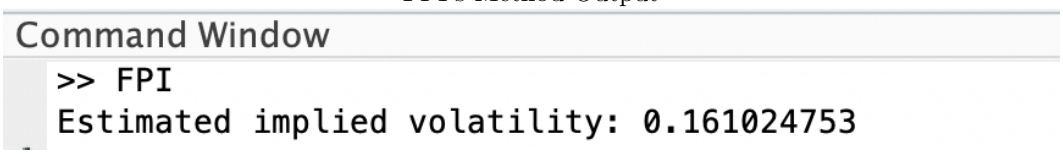
Function (FPI_call):

```

%normcdf function Required to download the Statistics and Machine Learning Toolbox.
function price = FPI_call(S, K, T, r, sigma)
    d1 = (log(S / K) + (r + 0.5 * sigma^2) * T) / (sigma * sqrt(T));
    d2 = d1 - sigma * sqrt(T);
    price = S * normcdf(d1) - K * exp(-r * T) * normcdf(d2);
end
```

Main Script:

```


% Parameters
S = 100;  % Current stock price
K = 95;   % Option strike price
T = 0.5;  % Time to maturity (in years)
r = 0.02; % Risk-free interest rate
P_market = 8;  % Market price of the call option

% Initial guess for sigma (implied volatility)
sigma_guess = 0.2;  % This is an arbitrary starting point

% Parameters for the iterative process
max_iterations = 100;
tolerance = 1e-6;

% Iterative adjustment (not standard fixed-point iteration!)
for i = 1:max_iterations
    price_guess = FPI_call(S, K, T, r, sigma_guess);
    error = P_market - price_guess;

    % Adjust sigma - Note this is an arbitrary adjustment and not based on fixed-point theory
    sigma_guess = sigma_guess + error * 0.01;  % Adjustment factor, this is quite arbitrary

    % Check if the error is within the tolerance
    if abs(error) < tolerance
        break;  % The estimate is good enough
    end
end

% Output the estimated implied volatility
fprintf('Estimated implied volatility: %.9f\n', sigma_guess);
```

FPI's Method Output



Discussion of FPI Method
Pros: Simplicity and Broad Applicability

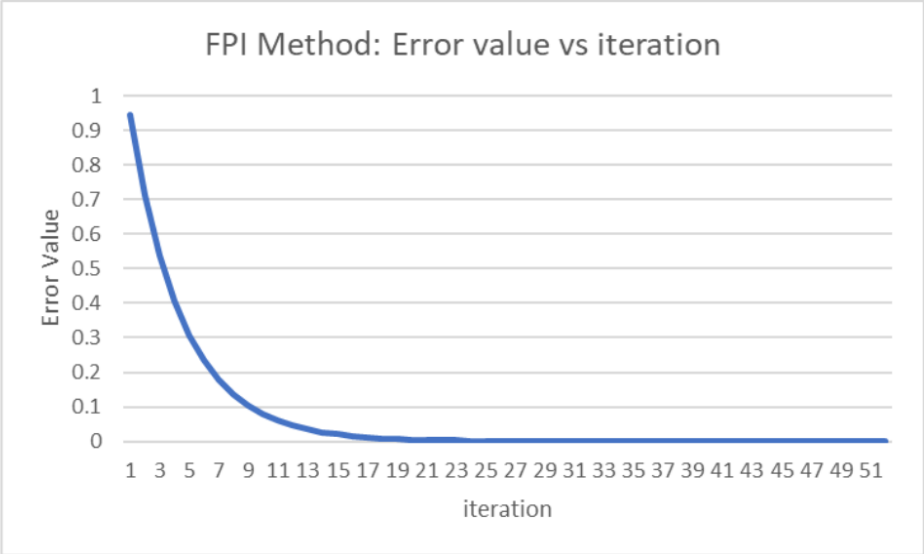
The Fixed-Point Iteration (FPI) method is simplistic and easy for implementation. It doesn't require the calculation of derivatives, making it less computationally intensive compared to Newton's Method. For tasks such as finding implied volatility, if one could suitably transform the Black-Scholes equation, the FPI method might provide a straightforward iterative mechanism.

If the function can be reformulated to meet the requirements of a fixed-point problem, then this method would be feasible, extending its range of application broadly.

Cons: Convergence Conditions and Rate:

The FPI method's major drawback is its dependency on the convergence conditions. The method requires that g(x) be a contraction mapping in the interval of interest to ensure convergence to a unique fixed point. Specifically, for the method to converge, it is necessary thatg(x)¡1 throughout the interval being considered.

The FPI method typically has linear convergence, which is slower compared to the quadratic convergence of Newton's Method or the convergence of the Secant Method. This slower rate of convergence can be a significant drawback in financial modeling where quick and accurate calculations are paramount.



Number of iteration to reach accuracy within 1e-6: **52 iterations**

6 Conclusion

The exploration of the Bisection Method, Newton's Method, Secant Method, and Fixed-Point Iteration in calculating implied volatility has provided valuable insights into the practical applications of numerical analysis in finance. Each method, with its unique approach and computational requirements, offers distinct advantages and limitations.

The Bisection Method, renowned for its reliability and simplicity, demonstrates consistent performance, albeit at the expense of computational speed. Newton's Method, with its rapid convergence, stands out in terms of efficiency, though it requires a good initial guess and the computation of derivatives. The Secant Method, eliminating the need for derivatives, shows versatility and speed but may lack the robustness of Newton's Method. Lastly, Fixed-Point Iteration, offering an intuitive approach, highlights the diversity of numerical techniques, though it demands careful formulation and may not always converge.

In conclusion, the choice of method for calculating implied volatility in the Black-Scholes model depends on the specific requirements of the financial analysis, including the desired balance between accuracy, speed, and computational resources. This study underscores the significance of numerical methods in financial decision-making, providing a foundation for further research and application in the dynamic environment of financial markets.

References:

1. https://www.investopedia.com/terms/b/blackscholes.asp
2. https://www.math.byu.edu/ matt/byu/math410_2007fall/projects/JonathanSmith.pdf
3. https://quantpy.com.au/black-scholes-model/implied-volatility-with-the-newton-raphson-method/

7 Appendix

To find the derivatives of $N(d_1)$ and $N(d_2)$ with respect to σ, we need to apply the chain rule. The cumulative distribution function (CDF) of the standard normal distribution $N(x)$ is commonly used in the Black-Scholes formula. The chain rule states that if $y = f(g(x))$, then $dy/dx = f'(g(x)) \cdot g'(x)$.

Let's consider:

1. **Derivative of $N(d_1)$ with respect to σ:**

$$\frac{d}{d\sigma}N(d_1) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}d_1^2} \cdot \frac{d}{d\sigma}d_1$$

2. **Derivative of $N(d_2)$ with respect to σ:**

$$\frac{d}{d\sigma}N(d_2) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}d_2^2} \cdot \frac{d}{d\sigma}d_2$$

Now, let's find the derivatives $d_1/d\sigma$ and $d_2/d\sigma$:

3. **Derivative of d_1 with respect to σ:**

$$\frac{d}{d\sigma}d_1 = \frac{1}{\sigma\sqrt{T}} + \frac{1}{2}d_1$$

4. **Derivative of d_2 with respect to σ:**

$$\frac{d}{d\sigma}d_2 = \frac{1}{\sigma\sqrt{T}} + \frac{1}{2}d_2$$

Now, combining these results, we get:

$$\frac{d}{d\sigma}N(d_1) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}d_1^2} \cdot \left(\frac{1}{\sigma\sqrt{T}} + \frac{1}{2}d_1\right)$$

$$\frac{d}{d\sigma}N(d_2) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}d_2^2} \cdot \left(\frac{1}{\sigma\sqrt{T}} + \frac{1}{2}d_2\right)$$

Now, for $f'(\sigma)$ in the Newton's method update, we use the fact that:

$$f'(\sigma) = S \cdot \sqrt{T} \cdot \frac{d}{d\sigma}N(d_1)$$

Substituting the expression for $\frac{d}{d\sigma}N(d_1)$, we get:

$$f'(\sigma) = S \cdot \sqrt{T} \cdot \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}d_1^2} \cdot \left(\frac{1}{\sigma\sqrt{T}} + \frac{1}{2}d_1\right)$$

After simplifying, you get $f'(\sigma) = S \cdot \sqrt{T} \cdot \mathit{stats.norm.pdf}(d_1)$. The multiplication by $S \cdot \sqrt{T}$ is included to account for the terms present in the Black-Scholes formula.