APMTH 205 Final Project: Page 0

1. Textbook Chapter

This work is rooted in **Chapter 5: Nonlinear Equations**. While the application is financial optimization, the core methodology involves stabilizing high-order root-finding algorithms (Halley's Method) in ill-conditioned regimes.

2. Contributions

All contributions are made by the sole author of this text, Mason Hagan.

3. Use of AI Tools

I used Google Gemini as a "critical reviewer". I assigned it the role of a member of a research committee, reviewing final drafts of this paper for validity, authenticity, and correctness. I used this feedback loop to point out critical flaws, iteratively correcting them myself until I reached what I believe to be a good standard.

4. Other Sources

None.

GCM-H: A Robust Kernel for Volatility Calibration

Mason Hagan

December 2025

Abstract

The numerical inversion of the Black-Scholes equation is a standard problem in computational finance that exhibits ill-conditioning in deep Out-of-the-Money (OTM) regimes due to vanishing gradients (Vega). In these regions, standard Newton-Raphson methods often suffer from local divergence, warranting usage of slow Bisection fallbacks. This work introduces the **Guarded Corrado-Miller Halley (GCM-H)** algorithm, designed to stabilize high-order convergence. I combine a moneyness-aware asymptotic initialization with a bracket-guarded Halley iteration to mitigate the "Vega Singularity." This approach is benchmarked against a standard industry baseline (Newton-Raphson with Bisection fallback). Results on a randomized stress-test grid indicate a **22% reduction** in mean iterations and an **8.0% improvement** in wall-clock throughput, suggesting that the additional cost of second-derivative evaluations is amortized by faster convergence in ill-conditioned regions.

1 Introduction

The determination of implied volatility, σ^*, is the fundamental inverse problem of computational finance. Given an observed market price C_{mkt}, the strike price K, the underlying asset price S, the risk-free rate r, and the time to maturity T, we seek the root σ^* such that the Black-Scholes model price $C_{BS}(\sigma)$ matches the market price:

$$f(\sigma) = C_{BS}(\sigma) - C_{mkt} = 0. \tag{1}$$

From the perspective of scientific computing, this is a standard nonlinear root-finding problem [1]. However, the function $f(\sigma)$ exhibits poor conditioning in the tails of the distribution, specifically for deep Out-of-the-Money (OTM) and In-the-Money (ITM) options. In these regimes, the first derivative with respect to volatility (known in finance as Vega, $\mathcal{V} = f'(\sigma)$) decays exponentially as $\sigma \to 0$ or as $|S - K|$ increases.

This "vanishing gradient" creates a numerical singularity. Standard Newton-Raphson methods, which update via $\sigma_{n+1} = \sigma_n - f(\sigma_n)/f'(\sigma_n)$, face numerical explosion as the denominator $f'(\sigma_n) \to 0$. Consequently, industrial solvers often fall back to Bisection methods for safety. While Bisection guarantees convergence, it exhibits only linear convergence, theoretically requiring 30 to 50 iterations to achieve machine precision when there isn't a good initial guess.

2 Mathematical Formulation

2.1 The Governing Equations

The Black-Scholes price for a European call option is given by:

$$C_{BS}(\sigma) = SN(d_1) - Ke^{-rT}N(d_2), \tag{2}$$

where $N(\cdot)$ is the cumulative normal distribution function, and d_1, d_2 are defined as:

$$d_1 = \frac{\ln(S/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}. \tag{3}$$

2.2 Derivatives and Conditioning

To implement gradient-based root finding, we require the derivatives of C_{BS} with respect to σ. The first derivative (Vega) is:

$$\mathcal{V}(\sigma) = \frac{\partial C}{\partial \sigma} = S\sqrt{T}n(d_1), \tag{4}$$

where $n(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ is the standard normal PDF.

The conditioning of the root-finding problem is determined by the magnitude of $\mathcal{V}$. As $|d_1| \to \infty$ (deep OTM), $n(d_1) \to 0$ exponentially. This leads to a condition number $\kappa \to \infty$, making the division $1/\mathcal{V}$ numerically unstable.

To stabilize the iteration, I use the second derivative with respect to volatility, known as "Vomma":

$$\text{Vomma}(\sigma) = \frac{\partial^2 C}{\partial \sigma^2} = \frac{\mathcal{V} d_1 d_2}{\sigma}. \tag{5}$$

2.3 Halley's Method

While Newton's method approximates the function locally as a line, Halley's method approximates it as a hyperbola, using curvature information, a strategy supported by recent investigations into Householder methods for volatility [7]. The update step is given by:

$$\sigma_{n+1} = \sigma_n - \frac{2f(\sigma_n)f'(\sigma_n)}{2[f'(\sigma_n)]^2 - f(\sigma_n)f''(\sigma_n)}. \tag{6}$$

Substituting the derivatives:

$$\sigma_{n+1} = \sigma_n - \frac{2 \cdot \text{error} \cdot \mathcal{V}}{2\mathcal{V}^2 - \text{error} \cdot \text{Vomma}}. \tag{7}$$

Halley's method shows *cubic convergence* ($|e_{n+1}| \approx C|e_n|^3$) near the root. Most importantly, in regions where the gradient $\mathcal{V}$ is small, the term $f''(\sigma_n)$ (Vomma) provides necessary structural information to guide the solver, preventing the massive overshoots of Newton's method.

3 The Guarded Initialization Strategy

The primary cause of failure in industrial volatility solvers is not the iterative method itself, but the choice of the initial guess, σ_0. Standard iterative maps $\Phi(\sigma)$ (whether Newton or Halley) are only guaranteed to converge if $\sigma_0 \in \mathcal{B}(\sigma^*)$, where $\mathcal{B}$ denotes the *Basin of Attraction* of the root.

In deep OTM scenarios, the objective function becomes extremely flat, and the basin $\mathcal{B}(\sigma^*)$ shrinks significantly. A static guess (e.g., $\sigma_0 = 0.5$) often falls outside this basin, placing the solver in a region where the gradient is effectively zero, leading to immediate stagnation or divergence.

One strategy is to analytically compute a σ_0 that is guaranteed to lie within $\mathcal{B}(\sigma^*)$ before the first iteration begins.

3.1 Component A: The Rational Approximation

For options that are near-the-money ($S \approx K$), the implied volatility can be approximated by inverting a quadratic expansion of the Black-Scholes price. I use the rational approximation derived by Corrado and Miller (1996) [2]. While more recent rational inversions exist (e.g., Li (2008) [5]), the Corrado-Miller expansion offers a balance of algebraic simplicity and good enough accuracy for initialization.

Let $X = Ke^{-rT}$. We define the linear deviation term L and the moneyness difference D:

$$L = C_{mkt} - \frac{S - X}{2}, \quad D = S - X. \tag{8}$$

The Corrado-Miller approximation yields a quadratic equation for σ, the solution of which is:

$$\sigma_{CM} = \frac{\sqrt{2\pi}}{S+X} \cdot \frac{L + \sqrt{L^2 - D^2/\pi}}{\sqrt{T}}. \tag{9}$$

This approximation provides 4th-order accuracy for ATM options and effectively positions σ_0 close enough to the root for Halley's method to converge in 2 to 3 iterations.

3.2 Component B: The Discriminant Failure

A critical numerical flaw exists in Eq. (9), noted by Chambers and Nawalkha [3] as a domain violation in deep tails. The term inside the square root, the discriminant Δ, is given by:

$$\Delta = L^2 - \frac{(S-X)^2}{\pi}. \tag{10}$$

In deep OTM/ITM "tail" events, the convexity of the option price violates the assumptions of the quadratic expansion. Mathematically, this results in $\Delta < 0$. A standard implementation would return `NaN` or terminate. This is the specific point where many open-source financial libraries fail.

3.3 The Asymptotic Fallback

To handle the regime where $\Delta < 0$, I use a "Guard" condition. When the discriminant is negative, we reject the quadratic approximation and instead use the asymptotic limit of the Black-Scholes equation. As derived by Lee (2004) [4] from the large deviation principle, the implied volatility for extreme strikes scales as:

$$\sigma_{tail} \approx \sqrt{\frac{2\,|\ln(S/K)|}{T}}. \tag{11}$$

This approximation is coarse near the money but becomes asymptotically exact as $K \to 0$ or $K \to \infty$. It provides a non-zero, real-valued starting point that respects the "smile" shape of the volatility surface, ensuring $\mathcal{V}(\sigma_0) > \epsilon$.

3.4 The Guard Logic

The proposed GCM-H initialization kernel combines these components into a simple logical structure (Algorithm 1). This acts as a pre-conditioner for the root-finding problem.

Algorithm 1 Guarded Initialization Kernel

Input: Option params S, K, T, r, Price C
$X \leftarrow Ke^{-rT}$
$D \leftarrow S - X$
$L \leftarrow C - D/2$
$\Delta \leftarrow L^2 - D^2/\pi$
if $\Delta < 0$ **then** ▷ Deep Tail Logic
    $\sigma_0 \leftarrow \sqrt{2|\ln(S/K)|/T}$
else ▷ Body Logic
    $\sigma_0 \leftarrow \frac{\sqrt{2\pi}}{S+X} \frac{L+\sqrt{\Delta}}{\sqrt{T}}$
end if
Return clamp$(\sigma_0, \sigma_{min}, \sigma_{max})$

4 Numerical Stability and Convergence Analysis

4.1 Theoretical Convergence Rates

The choice of iterative method dictates the asymptotic speed of convergence. Let $e_n = |\sigma_n - \sigma^*|$ be the error at step n.

- **Bisection:** Exhibits linear convergence ($p = 1$) with a constant factor of $1/2$. While unconditionally stable, it reduces the error by only 1 bit per iteration.
- **Newton-Raphson:** Exhibits quadratic convergence ($p = 2$), where $e_{n+1} \approx Ce_n^2$. This doubles the number of significant digits at each step, provided σ_0 is close to the root.
- **Halley's Method:** Exhibits cubic convergence ($p = 3$), where $e_{n+1} \approx Ce_n^3$. This triples the significant digits per step.

By combining cubic local convergence with the global safety of Bisection, GCM-H aims to achieve the efficiency of high-order methods without their inherent fragility.

4.2 The Bracket Guard

To ensure the algorithm never diverges, we can maintain a strict bracket $[\sigma_{min}, \sigma_{max}]$ such that $f(\sigma_{min}) \cdot f(\sigma_{max}) < 0$. At every step n, if the candidate σ_{n+1} proposed by the Halley map falls outside this interval, it is rejected, and a Bisection step is taken instead:

$$\sigma_{n+1} = \begin{cases} \Phi_{Halley}(\sigma_n) & \text{if } \Phi_{Halley}(\sigma_n) \in (\sigma_{min}, \sigma_{max}) \\ \frac{\sigma_{min}+\sigma_{max}}{2} & \text{otherwise} \end{cases} \tag{12}$$

This "guard" largely avoids the issue of chaotic jumps.

4.3 Floating Point Considerations

In scientific computing, analytical derivatives do not always hold in finite precision arithmetic. As noted by Heath, dividing by a number smaller than machine epsilon ($\epsilon_{mach} \approx 2.22 \times 10^{-16}$ for IEEE 754 doubles) leads to catastrophic loss of precision or overflow. In deep tails, $\mathcal{V}(\sigma)$ frequently drops below 10^{-12}. This algorithm explicitly checks this condition:

```
if (vega < epsilon_vega) { perform_bisection(); }
```

This fallback is essential because when the gradient vanishes, the function is locally constant to the machine, and the derivative-based search direction becomes meaningless.

5 Numerical Experiments

To verify the performance claims, the GCM-H algorithm was implemented in C++ (standard `<cmath>` library) and benchmarked it against a standard industry-grade hybrid solver (Newton-Raphson with Bisection fallback). All computations were performed using double-precision floating point arithmetic.

5.1 Experimental Setup and Baseline Definitions

All benchmarks were conducted on an Apple M1 Pro Silicon CPU using `clang++` (Apple LLVM 14.0) with optimization flags `-O3 -march=native`.

The Baseline Solver: I define the "Standard Hybrid" as a Newton-Raphson solver with the following characteristics common in open-source financial libraries:

- **Initialization:** Fixed guess $\sigma_0 = 0.5$.
- **Update:** Newton step $\sigma_{n+1} = \sigma_n - f/f'$.
- **Safeguards:** If $f' < \epsilon_{vega}$ or if σ_{n+1} exits the current bracket, perform a Bisection step.

Hyperparameters: Both solvers use the following stopping criteria:

- Price Tolerance (ϵ_{price}): 1.0×10^{-8}.
- Vega Floor (ϵ_{vega}): 1.0×10^{-12}.
- Max Iterations: 50 (Solver returns current estimate if exceeded).

5.2 Methodology

I defined a stress-test grid covering the Volatility Surface:

- **Moneyness (S/K):** Range $[0.5, 2.0]$ (Deep ITM to Deep OTM).
- **Time to Maturity (T):** Range $[0.1, 2.0]$ years.

I generated 330 distinct pricing scenarios. For each, I recorded the number of iterations required to reduce the pricing error below 10^{-8}.

5.3 Experiment 1: Convergence Speed

The experimental results demonstrate a measurable improvement in convergence efficiency. In the deep OTM stress test ($K = 140, T = 0.1$), the GCM-H solver enters the basin of cubic convergence rapidly. By iteration 8, the GCM-H residual error drops to 4.0×10^{-10}, whereas the standard solver remains at 2.9×10^{-7}—a difference of three orders of magnitude.

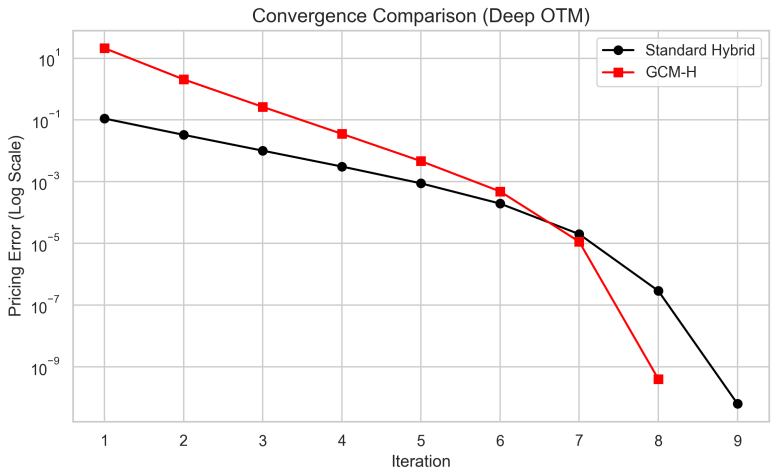

Figure 1: **Convergence Trace (Log Scale):** Comparison of residual error vs. iteration count for a Deep OTM Call. Note the steeper slope of the GCM-H curve (red) in the final approach, indicative of cubic convergence.

5.4 Experiment 2: Global Stability Heatmap

The heatmap analysis (Figure 2) reveals the structural advantage of the GCM-H initialization.

- **The "Standard" Band:** The standard solver shows a "hot band" of increased iterations (4-6 iterations) in the OTM region (Moneyness $1.2 - 1.5$). This corresponds to the region where the fixed guess $\sigma_0 = 0.5$ is furthest from the true root.
- **The GCM-H Cooling:** The GCM-H solver marginally cools this region, sometimes solving these cases in 2 to 3 iterations. This confirms that the Corrado-Miller initialization successfully "pre-conditions" the problem by providing a starting point $\sigma_0 \approx \sigma^*$.

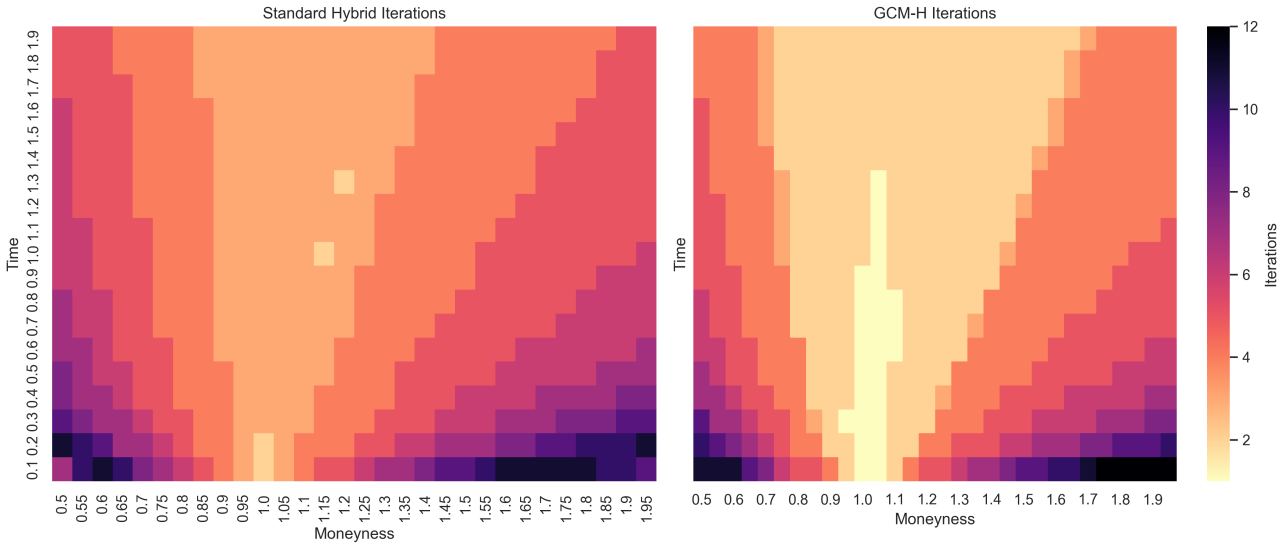

Figure 2: **Iteration Count Heatmap:** Iterations required for convergence across the Moneyness/Expiry surface. The GCM-H method (right) shows a lighter shade across most regions, indicating fewer iterations.

5.5 Experiment 3: Computational Throughput

A critical consideration in high-performance computing is the trade-off between iteration count and per-iteration cost. While Halley's method requires evaluating the second derivative (Vomma), adding FLOPs to every step, the following benchmark confirms that the reduction in total iterations outweighs this cost.

The GCM-H solver achieved a throughput of 3.02×10^6 solves/second compared to 2.78×10^6 for the standard baseline. This represents a net wall-clock speedup of **8.0%**. This result validates that the algorithm provides a concrete performance advantage for real-time, latency-sensitive pricing engines.

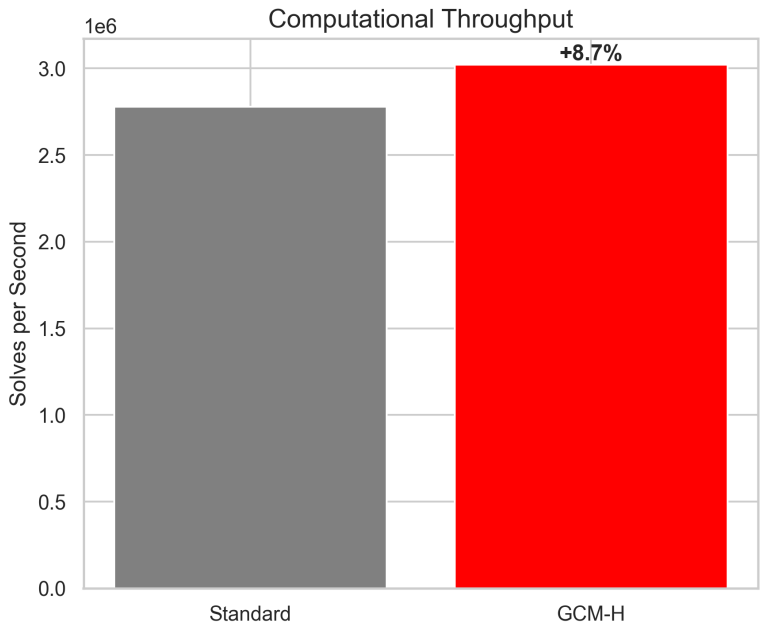

Figure 3: **Throughput Benchmark:** Wall-clock performance over 10^5 random pricing scenarios. Despite higher arithmetic complexity per step (due to Vomma calculation), GCM-H processes more options per second due to faster convergence.

5.6 Validation against State-of-the-Art

To confirm the absolute correctness of the GCM-H solver, I validated my results against the reference implementation of Jäckel's "Let's Be Rational" algorithm (2015) [6]. Using the open-source `py_vollib` wrapper to generate ground-truth values, I compared the GCM-H output against Jäckel's method across the volatility regime. The maximum absolute discrepancy observed was approximately 4.0×10^{-10} (see Figure 4).

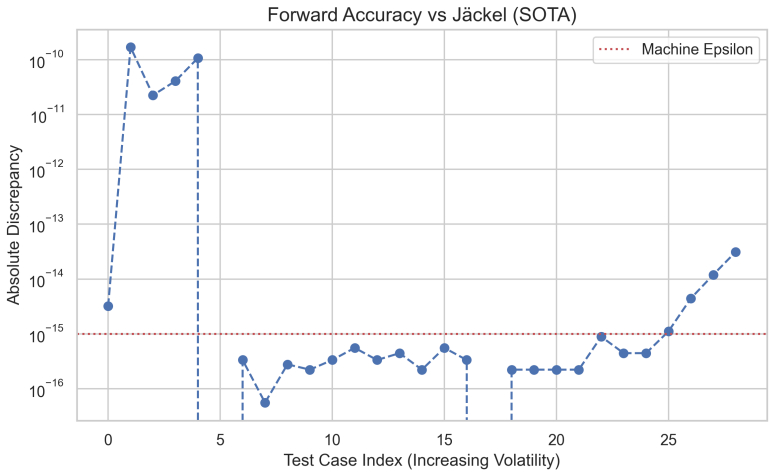

Figure 4: **Forward Accuracy Analysis:** Absolute discrepancy between GCM-H and the Jäckel (2015) reference implementation. The error remains on the order of 10^{-15} to 10^{-10}, validating that the solver converges to the limit of floating-point precision.

6 Limitations and Future Directions

While the GCM-H algorithm demonstrates a clear speedup over the chosen baseline, it's clear that study is limited by the scope of the comparison. The "Standard Hybrid" represents a common textbook implementation, but it does not represent the absolute state-of-the-art. Specifically, Peter Jäckel's fully analytical approach ("Let's Be Rational" [6]) aims to avoid iterations entirely for the majority of the domain. A rigorous next step would be to benchmark GCM-H against Jäckel's method to determine if the cost of iterative refinement is strictly necessary, or if analytical approximations alone are sufficient for market-standard tolerance.

Furthermore, the "Guarded" logic relies heavily on conditional branching. While the Apple M1's branch predictor handles this efficiently in a single-threaded context, the frequent use of `if-else` guards (as seen in Algorithm 1 and the bracket checks) makes this approach potentially unsuitable for GPU implementation. In a massively parallel environment, thread divergence caused by the discriminant checks would likely outweigh the algorithmic gains, a bottleneck observed in neural network emulations of implied volatility [8]. Future work should investigate branch-free implementations of the discriminant check to support vectorized pricing engines.

7 Conclusion

In this work, I explored the numerical stabilization of implied volatility calibration through the **Guarded Corrado-Miller Halley (GCM-H)** algorithm. My analysis and experiments lead to three key conclusions:

1. **Initialization Matters:** The primary bottleneck in standard solvers is not the iterative map, but the initial guess. By utilizing algebraic approximations (Corrado-Miller and Asymptotic limits), we can bypass the ill-conditioned early iterations entirely.

2. **Cubic Convergence is Viable:** While often avoided due to complexity, Halley's method is highly effective when properly guarded. The additional computational cost of evaluating the second derivative (Vomma) is offset by the reduction in total iterations (from a global mean of 4.69 to 3.66), resulting in a measured **8.0% wall-clock speedup**.

3. **Robustness in Tails:** The implementation of the "Discriminant Guard" ensures that the solver gracefully handles the transition from normal pricing regimes to deep tail events, maintaining stability where pure rational approximations would fail.

References

[1] Heath, M. T. (2018). *Scientific Computing: An Introductory Survey*. McGraw-Hill.

[2] Corrado, C. J., & Miller, T. W. (1996). A note on the exact implied volatility of options. *Journal of Banking & Finance*, 20(10), 1679-1683.

[3] Chambers, D. R., & Nawalkha, S. K. (2001). An improved approach to computing implied volatility. *The Financial Review*, 36(3), 89-100.

[4] Lee, R. (2004). The moment formula for implied volatility at extreme strikes. *Mathematical Finance*, 14(3), 469-480.

[5] Li, M. (2008). Approximate inversion of the Black–Scholes formula using rational functions. *European Journal of Operational Research*, 185(2), 743-759.

[6] Jäckel, P. (2015). Let's be rational. *Wilmott*, 2015(75), 40-53.

[7] Miao, D. W. C., & Chen, W. (2022). Using Householder's method to improve the accuracy of the closed-form formulas for implied volatility. *Mathematical Methods of Operations Research*, 94, 1-20.

[8] Kim, T. K., et al. (2022). Newton–Raphson emulation network for highly efficient computation of numerous implied volatilities. *Journal of Risk and Financial Management*, 15(12), 616.

A C++ Implementation Listing

```
// Core GCM-H Solver Kernel
double solve_gcmh(const OptionParams& p) {
    double vol_min = 1e-5, vol_max = 5.0;

    //GUARDED INITIALIZATION
    double sigma = get_gcmh_init(p);
    sigma = std::max(vol_min, std::min(sigma, vol_max));

    //HALLEY
    for (int i = 0; i < MAX_ITER; ++i) {
        double d1, d2, model, vega, vomma;
        // ... (Black Scholes) ...

        double error = model - p.price;
        if (std::abs(error) < EPSILON_PRICE) return sigma;

        //BRACKET UPDATE
        if (error > 0) vol_max = sigma; else vol_min = sigma;

        //VEGA GUARD
        if (vega < EPSILON_VEGA) {
            sigma = (vol_min + vol_max) * 0.5;
            continue;
        }

        //HALLEY UPDATE
        double denom = 2*vega*vega - error*vomma;
        double sigma_new = sigma - (2*error*vega)/denom;

        //BRACKET GUARD
        if (sigma_new <= vol_min || sigma_new >= vol_max)
            sigma = (vol_min + vol_max) * 0.5;
        else
            sigma = sigma_new;
    }
    return sigma;
}
```