Option Pricing under Rough Volatility (RFSV Model)

Gary Pai

September 21, 2026

1 Introduction

This document outlines the **Rough Fractional Stochastic Volatility (RFSV)** model framework based on the seminal paper *"Volatility is rough"* by Jim Gatheral, Thibault Jaisson, and Mathieu Rosenbaum (2014). It features both the analytical continuous-time mathematical definition and a practical, performant JavaScript implementation framework optimized using Typed Arrays for Monte Carlo simulation.

2 Model Specification Framework

The RFSV framework shifts away from standard Brownian diffusion models by driving log-volatility with a fractional Brownian motion containing a low Hurst parameter ($H < 1/2$). The risk-neutral simulation framework is structured beneath three coupled stochastic equations:

2.1 Log-Asset Price Process

The log-price of the underlying asset $Y_t = \log S_t$ satisfies the continuous semimartingale dynamic:

$$dY_t = \left(r - \frac{1}{2}\sigma_t^2 \right) dt + \sigma_t dW_t^S \tag{1}$$

where r is the risk-free interest rate and W_t^S is a standard Brownian motion under the pricing measure.

2.2 Log-Volatility Dynamic

The pathwise log-volatility $X_t = \log \sigma_t$ is modeled as a stationary fractional Ornstein-Uhlenbeck (fOU) process:

$$dX_t = \nu dW_t^H - \alpha(X_t - m)dt \tag{2}$$

where:

- W_t^H is a fractional Brownian motion (fBM) with a typical empirical regularity coefficient $H \approx 0.1$.
- ν defines the volatility of volatility (*vol-of-vol*).
- α represents the mean-reversion speed. As detailed in the paper, when looking at intra-horizon derivatives where $\alpha \ll 1/T$, the equation simplifies locally to a driftless fBM path: $X_t \approx X_0 + \nu W_t^H$.

2.3 Leverage Correlation Structure

To capture the classic equity market leverage effect and option smile skew, the asset price driver W_t^S and the underlying Brownian component of the volatility driver W_t are explicitly cross-correlated:

$$d\langle W^S, W \rangle_t = \rho dt \tag{3}$$

where $\rho \in [-1, 0]$ controls the asymmetrical skew amplitude of the implied volatility surface.

3 JavaScript Pseudo-Code Implementation

Below is the highly optimized JavaScript framework utilizing `Float64Array` allocations to secure memory locality and bypass engine garbage collection latency during intensive pricing loops.

```
/**
 * Rough Fractional Stochastic Volatility (RFSV) Option Pricing Algorithm
 * Based on Gatheral, Jaisson, & Rosenbaum (2014) fractional OU dynamics.
 */
function priceRoughVolOption(params) {
    const {
        S0,           // Initial asset price
        X0,           // Initial log-volatility log(sigma_0)
        H,            // Hurst parameter (typically around 0.1 for rough vol)
        nu,           // Volatility of volatility (Vol of Vol)
        alpha,        // Mean reversion speed (extremely small, often treated as 0)
        m,            // Long-term mean level of log-volatility
        rho,          // Leverage correlation parameter between asset and vol
        r,            // Risk-free interest rate
        K,            // Strike price
        T,            // Time to maturity
        N_steps,      // Number of discrete time steps
        N_paths       // Number of Monte Carlo simulation paths
    } = params;

    const dt = T / N_steps;
    const sqrt_dt = Math.sqrt(dt);

    //
        ================================================================

    // Step 1: Generate Covariance Matrix for Fractional Brownian Motion (fBM)
    // Based on paper equation (1.1): E[W_H(t) * W_H(s)] = 0.5 * (t^2H + s^2H - |t-s|^2H)
    //
        ================================================================

    let Sigma_fBM = Array.from({ length: N_steps }, () => new Float64Array(N_steps));

    for (let i = 0; i < N_steps; i++) {
        for (let j = 0; j < N_steps; j++) {
            const t_i = (i + 1) * dt;
```

```
            const t_j = (j + 1) * dt;
            Sigma_fBM[i][j] = 0.5 * (
                Math.pow(t_i, 2 * H) +
                Math.pow(t_j, 2 * H) -
                Math.pow(Math.abs(t_i - t_j), 2 * H)
            );
        }
    }

    //
        ================================================================

    // Step 2: Compute Cholesky Decomposition to generate the rough
        paths
    //
        ================================================================

    let L_fBM = choleskyDecomposition(Sigma_fBM, N_steps);

    //
        ================================================================

    // Step 3: Monte Carlo Path Simulation Main Loop
    //
        ================================================================

    let sumPayoffs = 0.0;

    for (let p = 0; p < N_paths; p++) {
        // Generate independent standard Gaussian vectors for the two
            paths
        let Z1 = generateStandardNormalVector(N_steps);
        let Z2 = generateStandardNormalVector(N_steps);

        // Multiply Cholesky matrix by uncorrelated noise to get the
            rough fBM path
        let W_H = matrixVectorMultiply(L_fBM, Z1, N_steps);

        let Y_current = Math.log(S0); // Current log-asset price
        let X_current = X0;           // Current log-volatility

        for (let i = 0; i < N_steps; i++) {
            // Compute the rough fBM increment (dW_H) for this step
            let dW_H = (i === 0) ? W_H[i] : (W_H[i] - W_H[i - 1]);

            // Uncorrelated spot driver component
            let dz2 = Z2[i] * sqrt_dt;

            // Construct the correlated stock Brownian motion increment
                (dW_S)
            // correlated back to the underlying volatility driver (Z1)
                via rho
            let dW_S = rho * (Z1[i] * sqrt_dt) + Math.sqrt(1 - rho *
                rho) * dz2;

            // Update fractional OU log-volatility using paper equation
                (3.3)
            // dX_t = nu * dW_H - alpha * (X_t - m) * dt
```

```
            let dX = nu * dW_H - alpha * (X_current - m) * dt;
            X_current += dX;

            // Transform back to physical volatility: sigma = exp(X)
            let sigma = Math.exp(X_current);

            // Update log-asset price: dY_t = (r - 0.5 * sigma^2) * dt
               + sigma * dW_S
            let dY = (r - 0.5 * sigma * sigma) * dt + sigma * dW_S;
            Y_current += dY;
        }

        // Step 4: Evaluate terminal payoff (Vanilla European Call
           Option)
        let ST = Math.exp(Y_current);
        let payoff = Math.max(ST - K, 0);
        sumPayoffs += payoff;
    }

    //
       ================================================================

    // Step 5: Risk-Neutral Discounting
    //
       ================================================================

    let optionPrice = Math.exp(-r * T) * (sumPayoffs / N_paths);
    return optionPrice;
}
```

Listing 1: RFSV European Option Pricing Algorithm