# Group: 6487 - Stochastic Modeling GWP 1

Shivansh Kumar, Viacheslav Karpov, Nikita Berezin

July 19, 2024

# 1.A Calibrating Heston (1993) Model Via Lewis (2001) Appraoch

As Our client is looking for a very short maturity for her derivative (around 15 days). we are going to calibrate a Classic Heston (1993) model (without jumps) to the observed market prices for both call and put options. we will use the Lewis (2001) approach with a regular MSE error function. We are considering a constant annual risk-rate of 1.50%. And We are also assuming 1 year have 250 trading days.

### 0.0.1 Heston (1993) Characteristic Function

We are going to start with a major ingredient for Fourier transform methods such as Lewis (2001) is knowledge of the characteristic function for the underlying process. We have used here the closed-form expression from the original Heston (1993)[1] paper or Gatheral (2006)[5] of the derivation of this characteristic function.

The characteristic function of the Heston (1993)[1] model is given by:

$$\varphi^H(u,T) = e^{H_1(u,T)+H_2(u,T)\nu_0}$$

where

$$H_1(u,T) \equiv r_0 uiT + \frac{c_1}{\sigma_\nu^2}\left\{(\kappa_\nu - \rho\sigma_\nu ui + c_2)T - 2log\left[\frac{1-c_3e^{c_2T}}{1-c_3}\right]\right\}$$

$$H_2(u,T) \equiv \frac{\kappa_\nu - \rho\sigma_\nu ui + c_2}{\sigma_\nu^2}\left[\frac{1-e^{c_2T}}{1-c_3e^{c_2T}}\right]$$

$$c_1 \equiv \kappa_\nu\theta_\nu$$

$$c_2 \equiv -\sqrt{(\rho\sigma_\nu ui - \kappa_\nu)^2 - \sigma_\nu^2(-ui-u^2)}$$

$$c_3 \equiv \frac{\kappa_\nu - \rho\sigma_\nu ui + c_2}{\kappa_\nu - \rho\sigma_\nu ui - c_2}$$

we have created a function in Python that simplifies its calculations every time:

### 0.0.2 Integral Value in Lewis (2001)

We also need to get a value for the integral in Lewis (2001):

$$C_0 = S_0 - \frac{\sqrt{S_0K}e^{-rT}}{\pi}\int_0^\infty \mathbf{Re}[e^{izk}\varphi(z-i/2)]\frac{dz}{z^2+1/4}$$

we have used this expression for the integral by changing the expression for the characteristic function.

### 0.0.3 Calculating the value of the integral and call & Put value

After that, we needed to numerically compute the value of the aforementioned integral. we used the quadrature method (quad) which is included in the scipy package. [14]

After this, we used some pre-defined parameters to check whether our pricing functions were working well or not.
we got value:

Heston (1993) Call Option Value: $ 5.7578
Heston (1993) put Option Value: $ 3.7777

For Put options pricing we have simply used the Put-call Parity i.e Put = Call + K * $e^{-r*t}$ - $S_0$
Next, we move into the most important part which is Heston model calibration.

### 0.0.4 Heston Model Calibration

In this section we are going to fully calibrate the Heston model with real data, we will first process our options data according to our requirement, Let's pre-process our data in the next section.

### Options Data Pre-processing

In the pre-processing section we loaded the data, Used the Skimpy package (https://pypi.org/project/skimpy/) to get all the necessary summary statistics about our data frame at one place, After that as Our stock SM Energy Company is currently trading at $232.90, so that will be the value of S0, After that we filtered our data frame to include on data which have Days to maturity of 15 days as that is our requirement for calibration. Next, we will add time left to maturity and a constant risk-free rate to our data frame.

After that, we started the calibration process.

### Calibration Process

Now this is our main section of the calibration process, we also need to define some additional functions to optimize the model parameters so that they work well on observed data.

First, we will introduce a function that will evaluate the error the model makes with respect to observed data given certain parameters. As usual, we will rely on a mean squared error (MSE) function. We will also define some initial values for the calibration parameters:

$$min \frac{1}{N} \sum_{n=1}^{N} \left(C_n^* - C_n^{Model}(\alpha)\right)\right)^2$$

**Algorithm 1** H93 Error Function

**procedure** H93_ERROR_FUNCTION(p0)
  **Input:**
    $p0$: A set of parameters including $\kappa_v$, $\theta_v$, $\sigma_v$, $\rho$, and $v0$
  **Extract parameters from** $p0$**:**
    $\kappa_v, \theta_v, \sigma_v, \rho, v0 \leftarrow p0$
  **Step 1: Parameter Validation**
  **if** $\kappa_v < 0.0$ or $\theta_v < 0.005$ or $\sigma_v < 0.0$ or $\rho < -1.0$ or $\rho > 1.0$ **then**
    **return** $500.0$
  **end if**
  **if** $2 \times \kappa_v \times \theta_v < \sigma_v^2$ **then**
    **return** $500.0$
  **end if**
  Initialize empty lists $se$ and $se2$
  **Step 2: Calculate Squared Errors for Each Option**
  **for** each $option$ in $options$ **do**
    **if** $option["Type"]$ is "C" **then** ▷ Call option
      $model\_value \leftarrow H93\_call\_value(S0, option["Strike"], option["T"], option["r"], \kappa_v, \theta_v, \sigma_v, \rho, v0)$
      Append $(model\_value - option["Price"])^2$ to $se$
    **end if**
    **if** $option["Type"]$ is "P" **then** ▷ Put option
      $model\_value \leftarrow H93\_call\_value(S0, option["Strike"], option["T"], option["r"], \kappa_v, \theta_v, \sigma_v, \rho, v0)$
      Append $(model\_value - option["Price"])^2$ to $se2$
    **end if**
  **end for**
  **Step 3: Calculate Mean Squared Errors**
  $MSE \leftarrow \frac{\sum se}{len(se)}$
  $MSE2 \leftarrow \frac{\sum se2}{len(se2)}$
  Update $min\_MSE \leftarrow \min(min\_MSE, MSE)$
  Update $min\_MSE2 \leftarrow \min(min\_MSE2, MSE2)$
  **Step 4: Print Status Every 100 Iterations**
  **if** $i\%100 == 0$ **then**
    Print iteration number $i$, parameters $p0$, Call MSE $MSE$, Min Call MSE $min\_MSE$
    Print iteration number $i$, parameters $p0$, Put MSE $MSE2$, Min Put MSE $min\_MSE2$
  **end if**
  Increment $i$
  **return** $MSE + MSE2$
**end procedure**

Next, we needed a function that performs the optimization process. In other words, it optimizes the model parameters so as to minimize the error function with respect to market data. We will do this in 2 steps in order to look for faster convergence of the prices to market quotes. First, we will use the brute function of Scipy [14], which allows the calibration to focus on the most sensible ranges. Once these are declared, we can dig deeper into the specific regions and get the actual parameters more accurately with the fmin function.

**Algorithm 2** H93 Calibration Full

**procedure** $\mathrm{H93\_CALIBRATION\_FULL}$
  **Step 1: Initial Brute Force Search**
  Define initial parameter ranges:
  - $\kappa_v \in [2.5, 10.6, 5.0]$
  - $\theta_v \in [0.01, 0.041, 0.01]$
  - $\sigma_v \in [0.05, 0.251, 0.1]$
  - $\rho \in [-0.75, 0.01, 0.25]$
  - $v_0 \in [0.01, 0.031, 0.01]$

  Run brute force search over the parameter ranges to find initial guess $p_0$
  **Step 2: Local Convex Minimization**
  Refine initial guess $p_0$ using a local optimizer:
  - Set tolerance levels: xtol $= 0.000001$, ftol $= 0.000001$
  - Set iteration limits: maxiter $= 750$, maxfun $= 900$

  Run local optimization starting from $p_0$ to find optimal parameters opt
  **return** opt
**end procedure**

### 0.0.5 Results

Now that we have all the necessary ingredients, let's see how our calibration algorithm performed. For that, given the way we structured things before, we just need to call our $H93\_calibration\_full()$ function. This will give us each of the different outputs from calibration, including the values given to the different parameters in the model.

Now we have finally calibrated our parameters to market values.

The results from the calibration gave us the following values for the parameters in the Heston (1993) model:

$$\begin{aligned} \kappa_\nu &= 5.379 \\ \theta_\nu &= 0.086 \\ \sigma_\nu &= 0.000 \\ \rho &= -0.006 \\ \nu_0 &= 0.087 \end{aligned}$$

The next step will be simply using these parameters to price the option we want. but before that first, let's plot our results to check how our model calibration has worked. We can see in Figure 1 and Figure 2 how our model has performed.

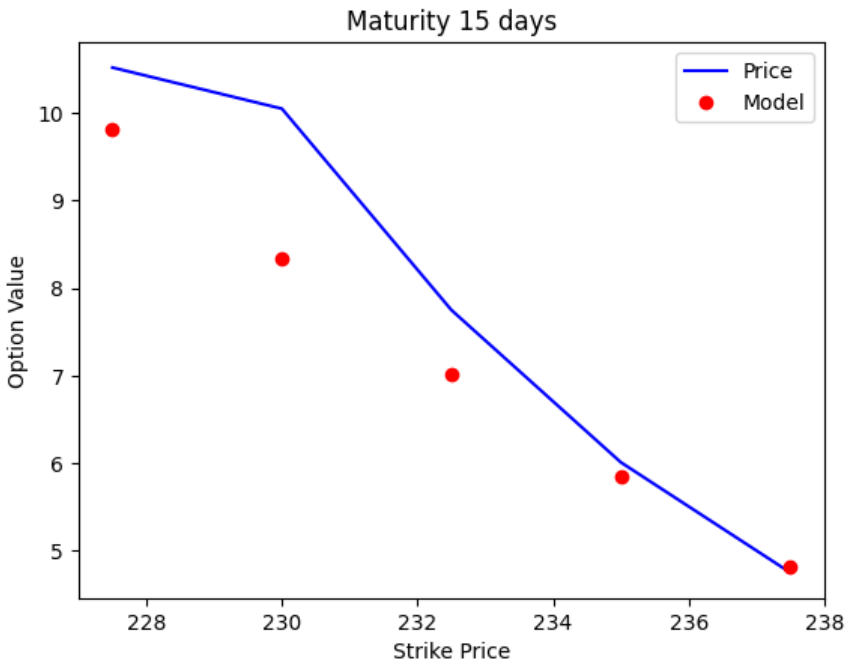


Figure 1: Call option Calibrated model

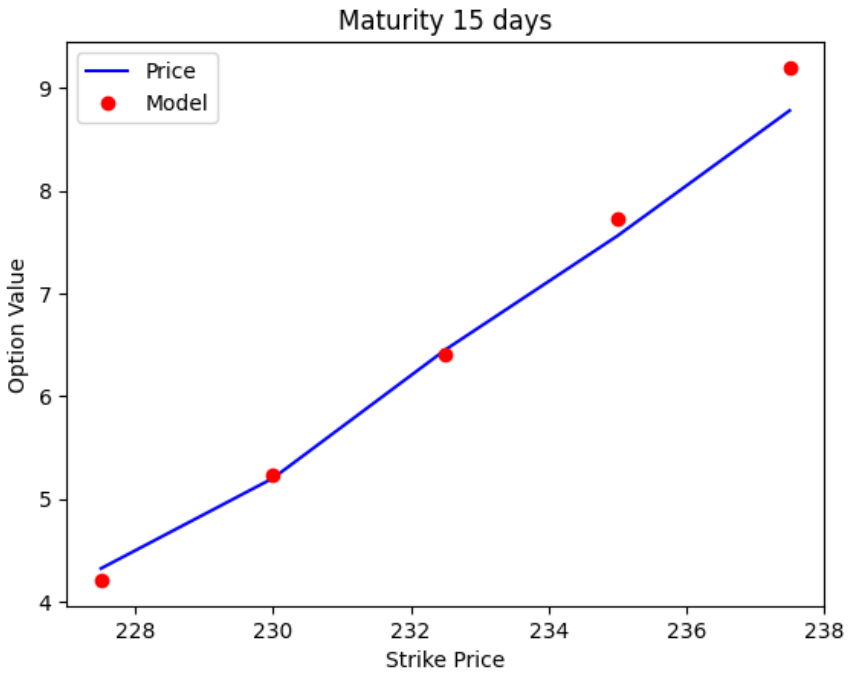


Figure 2: put option Calibrated model

# 1.b Calibrating Heston (1993) Model Via the Carr-Madan (1999) pricing approach

In this section we will use the Carr-Madan (1999) pricing approach. The Carr-Madan approach uses the Fast Fourier Transform (FFT) to price options, which brings several advantages comparig with the Lewis approach aplied in previous section (a):

- Efficiency: The FFT algorithm is computationally efficient with a time complexity of *O(NlogN)*, making it well-suited for the batch processing and pricing of many options quickly.

- Numerical Stability: By leveraging the FFT, the Carr-Madan method can handle a large number of integration points stably, leading to more accurate results.

- Simplicity: The method is relatively straightforward to implement using existing FFT libraries. Moreover it is well-suited for pricing exotic options and payoffs that can be expressed in terms of Fourier transforms.

In order to apply this approach we will use FFT to the integral in the call option price derived by Carr and Madan (1999):

$$C_0 = \frac{e^{-\alpha\kappa}}{\pi} \int_0^\infty e^{-i\nu\kappa} \frac{e^{-rT}\varphi^{H93}(\nu - (\alpha+1)i, T)}{\alpha^2 + \alpha - \nu^2 + i(2\alpha+1)\nu} d\nu$$

We will use provided market data to calibrate parameters of the Heston model.

### 0.0.6 Calibration Process

The calibration process for the Heston model aims to determine the optimal parameters for the model by minimizing the Mean Square Error (MSE) between market prices and model prices. Initially, a placeholder for the H93_value_FFT function, which is responsible for calculating option prices using the Heston model, is defined based on the step a. The objective function is then created to compute MSE. This function iterates through market data, computes model prices for given market conditions and parameters, and calculates the squared errors between these model prices and actual market prices. The RMSE, a measure of the average discrepancy between model and market prices, is then derived from these squared errors.

Market data, comprising columns for initial stock price ($S_0$), strike price ($K$), time to maturity ($T$), risk-free rate ($r$), option type, and market price, is structured based on a `pandas` DataFrame. An initial guess for the parameters of the Heston model, including the speed of mean reversion ($\kappa_v$), long-term variance ($\theta_v$), volatility of variance ($\sigma_v$), correlation between asset price and variance ($\rho$), and initial variance ($v_0$), is provided. Selected bounds ensure the parameters remain within usually observed ranges during the optimization process.

The `scipy.optimize.minimize` function is then employed to minimize the mean squared error (MSE), effectively calibrating the Heston model parameters to best fit the given market data. The result of the optimization yields the optimal values for $\kappa_v$, $\theta_v$, $\sigma_v$, $\rho$, and $v_0$, which are then extracted and printed. These optimized parameters are applied to the DataFrame to calculate option prices using the calibrated `H93_value_FFT` function.

As a result we have calibrated parameters of our model to market option prices.
The results of this calibration provided us with the following the parameters in the Heston (1993) model:

- $\kappa_\nu = 7.21$
- $\theta_\nu = 0.0001$
- $\sigma_\nu = 1.00$
- $\rho = 0.768$
- $v_0 = 0.01$

After that, we used the pandas apply method to compute model prices for each row in the DataFrame, creating a new column, 'Model,' in our DataFrame, which contains these calculated prices. The final DataFrame, now inclusive of both market prices and model prices calculated with the optimized parameters, provides a comprehensive view of the calibration accuracy and effectiveness.

The last step involves plotting a scatter plot to visually inspect how closely the modeled option prices, determined based on our function with calibrated parameters, fit the market data.

### 0.0.7 Results

Analyzing the graphs above, we can conclude that the parameters found using the calibration process fit the actual market prices relatively well. The estimated errors are: MSE $= 0.48$ and RMSE $\approx 0.7$. A visual inspection of the Put and Call graphs demonstrates that the model produces values close to the market data. The only noticeable difference is for the deep out-of-the-money call option.

| | Days to maturity | Strike | Price | Type | t | r | S0 | Model |
|---|---|---|---|---|---|---|---|---|
| **0** | 15 | 227.5 | 10.52 | C | 0.06 | 0.15 | 232.9 | 11.564879 |
| **1** | 15 | 230.0 | 10.05 | C | 0.06 | 0.15 | 232.9 | 9.738884 |
| **2** | 15 | 232.5 | 7.75 | C | 0.06 | 0.15 | 232.9 | 8.131079 |
| **3** | 15 | 235.0 | 6.01 | C | 0.06 | 0.15 | 232.9 | 6.809293 |
| **4** | 15 | 237.5 | 4.75 | C | 0.06 | 0.15 | 232.9 | 5.786555 |
| **15** | 15 | 227.5 | 4.32 | P | 0.06 | 0.15 | 232.9 | 4.126565 |
| **16** | 15 | 230.0 | 5.20 | P | 0.06 | 0.15 | 232.9 | 4.778171 |
| **17** | 15 | 232.5 | 6.45 | P | 0.06 | 0.15 | 232.9 | 5.647967 |
| **18** | 15 | 235.0 | 7.56 | P | 0.06 | 0.15 | 232.9 | 6.803782 |
| **19** | 15 | 237.5 | 8.78 | P | 0.06 | 0.15 | 232.9 | 8.258645 |

Figure 3: Resulting data frame with market data and model prices

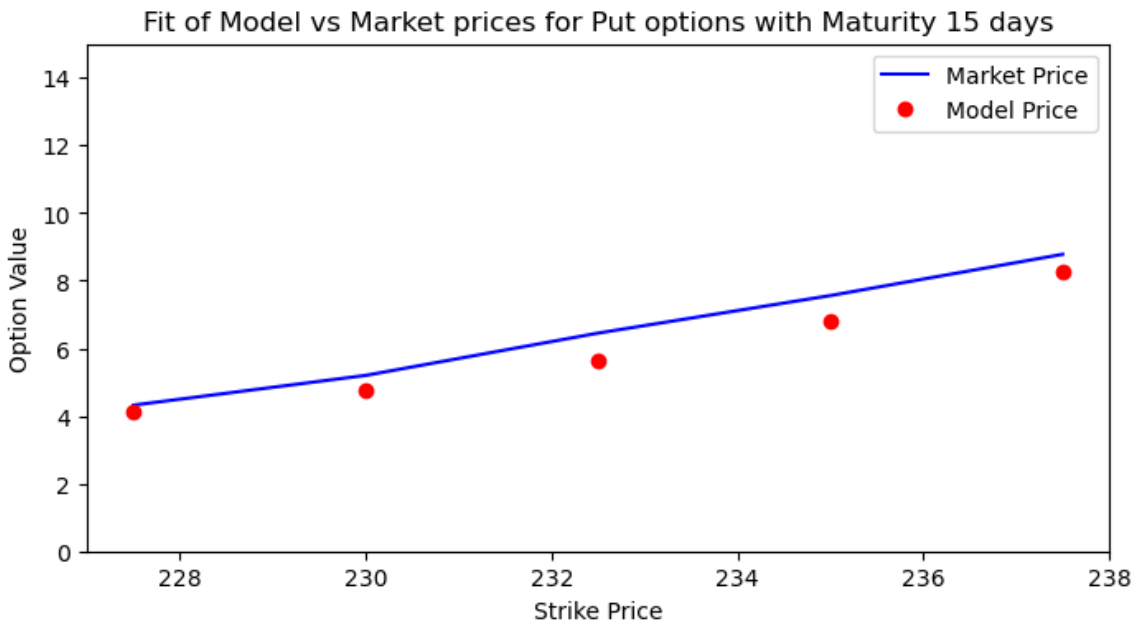


Figure 4: Call options

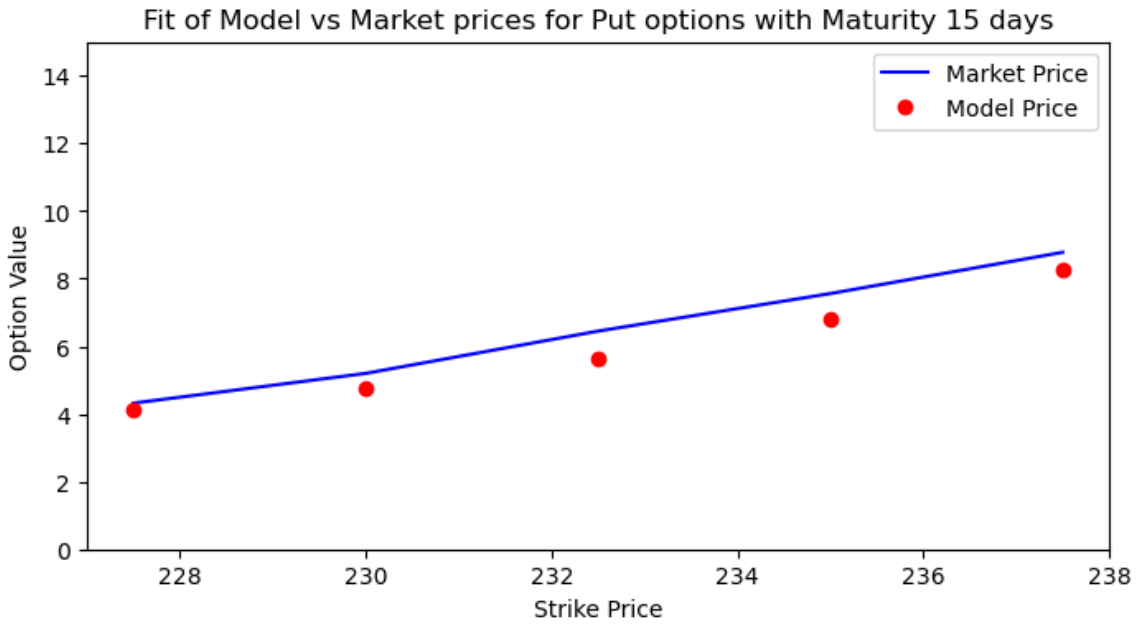


Figure 5: Put options

## 1.C Using Monte-Carlo simulations over Heston model for Asian call option pricing

Asian options, being path-dependent derivatives, present unique challenges in valuation. Asian options are path-dependent options whose payoff depends on the average price of the underlying asset over a specified period (Wilmott, 2006 [2]). For an arithmetic average Asian call option, the payoff at maturity $T$ is:

$$\text{Payoff} = \max(\bar{S} - K, 0)$$

Where $\bar{S}$ is the arithmetic average of the asset price over the option's life, and $K$ is the strike price.

### 0.0.8 The Heston Model

The Heston model (Heston, 1993 [1]) is defined by the following stochastic differential equations (SDEs):

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$$
$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t^v$$

Where: $S_t$ is the asset price at time $t$ $v_t$ is the variance at time $t$ $\mu$ is the drift of the asset $\kappa$ is the rate of mean reversion of variance $\theta$ is the long-term mean of variance $\sigma$ is the volatility of variance $\rho$ is the correlation between the asset price and variance processes $W_t^S$ and $W_t^v$ are standard Brownian motions

### 0.0.9 Discretization scheme for Monte-Carlo simulation

To simulate paths under the Heston model, we use the Euler-Maruyama discretization scheme (Kloeden & Platen, 1992 [3]). For a time step $\Delta t$, the discretized equations are:

$$S_{t+\Delta t} = S_t + rS_t\Delta t + \sqrt{v_t} S_t \sqrt{\Delta t} Z^S$$
$$v_{t+\Delta t} = v_t + \kappa(\theta - v_t)\Delta t + \sigma\sqrt{v_t}\sqrt{\Delta t} Z^v$$

Where $Z^S$ and $Z^v$ are standard normal random variables with correlation $\rho$.

### 0.0.10 Monte-Carlo simulations algorithm

Then, the Monte Carlo estimate for the Asian call option price is:

$$C_{\text{Asian}} \approx e^{-rT} \frac{1}{N} \sum_{j=1}^{N} \max(\bar{S}^j - K, 0)$$

Where $\bar{S}^j$ is the average price for the $j$-th simulated path.

**Algorithm 3** Monte Carlo Simulation for Asian Option Pricing

**Input:** Time to maturity $T$, number of steps $M$, number of simulations $N$, initial stock price $S_0$, initial variance $v_0$, strike price $K$
**Output:** Option price
Divide the time to maturity $T$ into $M$ equal steps: $\Delta t = T/M$
**for** each of $N$ simulations **do**
    Initialize $S_0$ and $v_0$
    **for** each time step $i = 1$ to $M$ **do**
        Generate correlated normal random variables $Z^S$ and $Z^v$
        Update $S_i$ and $v_i$ using the discretized equations
        Ensure $v_i$ remains non-negative (e.g., by using $\max(v_i, 0)$)
    **end for**
    Calculate the average price: $\bar{S} = \frac{1}{M+1} \sum_{i=0}^{M} S_i$
    Calculate the option payoff: $\max(\bar{S} - K, 0)$ for a call option
**end for**

### 0.0.11 Bias-Variance tradeoff

The choice of $M$ (time steps) and $N$ (number of simulations) involves a tradeoff:

Larger $M$ reduces discretization bias but increases computational cost. Larger $N$ reduces Monte Carlo error but increases computational cost.

The optimal choice depends on the specific option parameters and required accuracy.

### 0.0.12 Results

The simulation results stabilize after around 10,000 simulations and fluctuate around a fair price of $4.53. Charging an additional 4% fee will increase the effective option price up to $4.71.

# 2.A Calibration of Heston model with jumps (Bates, 1996) using Lewis (2001) approach

Now we assume an existence of price jumps in the underlying valuation and calibrate the Bates (1996) model to observed market prices for call and put options. We use the Lewis (2001) approach with a regular Mean Squared Error (MSE) function for optimization. The calibration is performed for options with a prolonged 60-day maturity contrasting with a previous 15-days one.

### 0.0.13 Bates (1996) Model

The Bates model (Bates, 1996 [8]) extends the Heston stochastic volatility model by incorporating jumps in the asset price process. The model is described by the following stochastic differential equations:

$$dS_t = (r - \lambda\mu_J)S_t dt + \sqrt{v_t}S_t dW_t^S + J_t dN_t$$
$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t}dW_t^v$$

where $S_t$ is the asset price, $v_t$ is the variance, $r$ is the risk-free rate, $\lambda$ is the jump intensity, $\mu_J$ is the expected jump size, $J_t$ is the jump size (typically log-normally distributed), $N_t$ is a Poisson process with intensity $\lambda$, and other parameters are as in the original Heston model.

### 0.0.14 Lewis (2001) Approach

Lewis provided a semi-analytical solution for option pricing under stochastic volatility models with jumps (Lewis, 2001 [13]). The call option price is given by:

$$C_0 = S - Ke^{-rT} + \frac{Ke^{-rT}}{\pi}\int_0^\infty \text{Re}\left[\frac{e^{-ik\ln(K/S)}\phi(k-i,T)}{ik\phi(-i,T)}\right]dk$$

Where $\phi(u,T)$ is the characteristic function of the log-price process.

### 0.0.15 Calibration Process

We use the Mean Squared Error (MSE) as our objective function:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(C_{\text{model},i} - C_{\text{market},i})^2$$

Where $N$ is the number of options, $C_{\text{model},i}$ is the model price, and $C_{\text{market},i}$ is the market price for the $i$-th option.

### 0.0.16 Results

Optimal parameters for Bates' (1996) jump component (Merton (1976)) with Lewis (2001) approach are

$$\lambda = 0.0000825$$
$$\mu = -0.2409$$
$$\delta = 3.704$$

along with Heston pre-calibrated parameters of

$$\kappa_\nu = 5.379$$
$$\theta_\nu = 0.086$$
$$\sigma_\nu = 0.000$$
$$\rho = -0.006$$
$$\nu_0 = 0.087$$

**Algorithm 4** Bates Model Jump Component Calibration with Lewis Approach

**procedure** BATES_JUMP_CALIBRATION_FULL
  **Step 1: Set Previously Calibrated Heston Parameters**
  $\kappa_\nu \leftarrow 5.379$
  $\theta_\nu \leftarrow 0.086$
  $\sigma_\nu \leftarrow 0.000$
  $\rho \leftarrow -0.006$
  $\nu_0 \leftarrow 0.087$
  **Step 2: Initial Brute Force Search for Jump Parameters**
  Define initial parameter ranges:
  - $\lambda \in [0.1, 2.1, 0.5]$
  - $\mu_J \in [-0.2, 0.2, 0.1]$
  - $\sigma_J \in [0.05, 0.301, 0.05]$

  Run brute force search over the jump parameter ranges to find initial guess $p_0$
  **Step 3: Local Convex Minimization**
  Refine initial guess $p_0$ using a local optimizer:
  - Set tolerance levels: xtol $= 0.000001$, ftol $= 0.000001$
  - Set iteration limits: maxiter $= 500$, maxfun $= 600$

  Run local optimization starting from $p_0$ to find optimal jump parameters opt
  **return** opt
**end procedure**

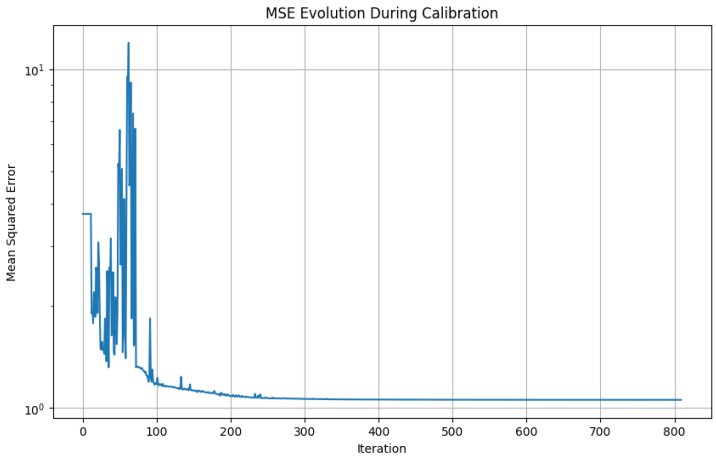


Figure 6: MSE evolution during Bates (1996) model calibration

# 2.B Calibrating Bates (1996) model using Carr Madan (1999) Approach

As our Client demanded that 60 days maturity would be better for her needs, we are going to use a different approach for calibration we are going to use the Carr-madan (1999)[7] approach to Bates (1996)[8]

As we have previously used Lewis's (2001) Approach to calibrate the models, this time we are going to use Carr-Madan(1999) to Bates (1996) for that we need Bates characteristic function which is simply the product of the characteristic function Heston model and Merton model Jump component.

As we have already discussed about the Heston model characteristic function in Step 1 a. We are going to start with the Merton Adjusted characteristic function where we will only include the jump component.

### 0.0.17 Merton (1976) Adjusted Characteristic Function (Only Jump Component)

In order to produce a stochastic volatility jump-diffusion model, we need to incorporate only the jump component of the Merton (1976)[4] characteristic function.

The adjusted (only jump) characteristic function of Merton (1976)[4] is therefore given by:

$$\varphi_0^{M76J}(u,T) = e^{\left(\left(iu\omega + \lambda(e^{iu\mu_j - u^2\delta^2/2} - 1)\right)T\right)}$$

where,

$$\omega = -\lambda\left(e^{\mu_j + \delta^2/2} - 1\right)$$

### 0.0.18 Bates (1996) Characteristic Function

This is the $\varphi^{B96}()$ characteristic function of the Bates model. which is the product of the Heston model[1] and the jump-diffusion part of the Merton model[4].

$$\varphi_0^{B96}(u,T) = \varphi_0^{H93}\varphi_0^{M76J}(u,T)$$

Now we can combine both previous characteristic functions to produce the one we are interested in.

### 0.0.19 Pricing via Carr and Madan (1999)

Now we are going to price call option via the Carr-madan (1999) model. we can apply FFT to the integral in the call option price derived by Carr and Madan:

$$C_0 = \frac{e^{-\alpha\kappa}}{\pi}\int_0^\infty e^{-i\nu\kappa}\frac{e^{-rT}\varphi^{B96}(\nu - (\alpha+1)i, T)}{\alpha^2 + \alpha - \nu^2 + i(2\alpha+1)\nu}d\nu$$

Here we are going to use a numerical FFT routine and as was the case with the Lewis (2001) approach, we basically have to adapt the characteristic function we are considering to be the Bates (1996) one.

We also quickly used the put-call parity to write a put-call pricing function, and then we used some general Parameters to check whether our functions were working well or not.

And the results were:

B96 Call option price via FFT: $ 8.9047
B96 Put option price via FFT: $ 4.0277

### 0.0.20 Options Data processing

after that, we moved into option data processing. We have already discussed data processing in Step 1 a. So we will not discuss anything more here but this time as our client has demanded 60 days of maturity, we are going to use data that has 60 days to maturity.

### 0.0.21 Calibrating Heston (1993) stochastic volatility model

we are going to start with $\varphi_0^{H93}$ which stands for the characteristic function of Heston (1993). Thus, our first task in calibrating the Bates (1996) model will be to calibrate the Heston (1993) model to observed market data. we have done something similar in step 1 a. So we are going to do the same thing again but with days to maturity of 60 days.

Again, we are going to use the same function that performs the optimization process. In other words, it optimizes the model parameters so as to minimize the error function with respect to market data.

we will use the same steps that is to use a brute function of Scipy[14], which allows the calibration to focus on the most sensible ranges. Once ranges are declared, we can dig deeper into the specific regions and get the actual parameters more accurately with the fmin function.

Now, let's see the performance of our calibration algorithm. For that, we just need to call our H93_calibration_full() function. This will give us each of the different outputs from calibration, including the values given to the different parameters in the model.

we have finally calibrated our parameters to market values. The results from this calibration give us the following values for the parameters in the Heston (1993) model:

$$\begin{aligned}\kappa_\nu &= 13.042\\ \theta_\nu &= 0.122\\ \sigma_\nu &= 0.000\\ \rho &= -0.003\\ \nu_0 &= 0.045\end{aligned}$$

The next step will be simply using these parameters to use in the final Bates model calibration. But before that let's calibrate the Jump component in the Bates model.

### 0.0.22 Calibrate jump component in Bates (1996)

Now we will move to calibrating the Merton model Jump diffusion component, we will start with an error function which will be looking at differences between market and model prices for the complete Bates (1996) model.

We are also going to use the same Carr-Madan (1999) pricing function of call and put options via bates (1996) model.

After that, we will create our function to calibrate the jump component of the model and after optimizing we will get our Jump component parameters.

$$\begin{aligned}\lambda &= 3.445\\ \mu &= -0.173\\ \delta &= 0.000\end{aligned}$$

### 0.0.23 Model vs Market prices after jump calibration

Before moving forward, we will check the performance of our jump-calibrated model. we will create a function that yields the model values under the different parameters obtained in the calibration process.

and here are the plot results in Figure 7

### 0.0.24 Full Bates (1996) model calibration

Now we are at our Final step i.e. Full calibration of the Bates (1996) model.

As we have already obtained our calibrated parameters from both the components—the stochastic volatility model $(\kappa_\nu, \theta_\nu, \sigma_\nu, \rho, \nu_0)$, and those from the jump component $(\lambda, \mu, \delta)$.

Hence, our $p_0$ is defined as:

$$p_0 = [\kappa_\nu, \theta_\nu, \sigma_\nu, \rho, \nu_0, \lambda, \mu, \delta]$$

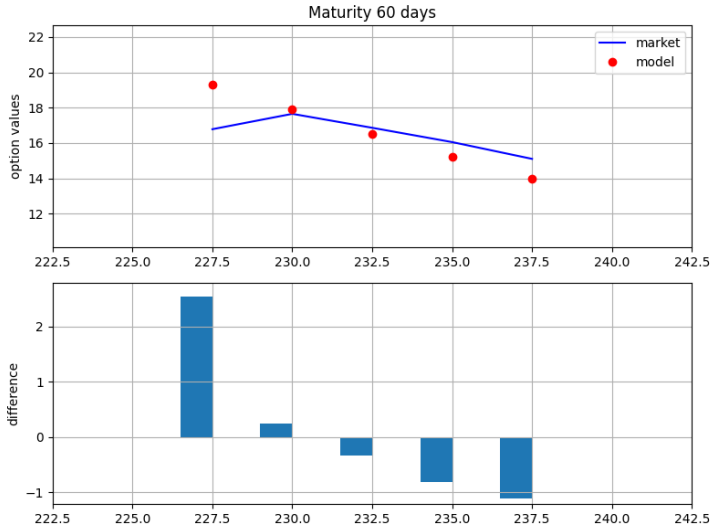


Figure 7: Model V/S Market Price Jump component

Now, we built up a B96_full_error_function that essentially takes these inputs and calculates the error function. Here we do not impose any penalties on the error function, as it solved an inherent problem of the jump-diffusion model that should be already been solved by using the initial inputs from the jump component calibration.

Next, we will use the classic calibration we have done before for other models and after calibration, we get the complete calibrated parameters :

$$\begin{aligned}
\kappa_\nu &= 13.042 \\
\theta_\nu &= 0.122 \\
\sigma_\nu &= 0.000 \\
\rho &= -0.003 \\
\nu_0 &= 0.045 \\
\lambda &= 3.159 \\
\mu &= -0.000 \\
\delta &= 0.001
\end{aligned}$$

### 0.0.25 Final Results Plotting

Again we will see market vs model prices in full calibration see the differences that our model produces using the parameters resulting from calibration, and compare those to observed market prices.

we can see in Figure 8 and Figure 9 how our model performed V/s the Market prices.

### 0.0.26 Bates Calibration Steps

To summarize all that we have done above we can conclude with the steps that we followed for the calibration process, we basically followed three step procedure:

- Calibration of basic stochastic volatility model (i.e calibrating Heston '93 [1])
- Calibration of Jump Component(i.e Calibrating Merton '76 [4] with only jump-diffusion)
- In this last step we took inputs as the results from the above two calibrations and fully calibrated the general model.

## 2.C Monte Carlo valuation of OTM Put option

In this section, we will price a Put option for "SM" with 70 days to maturity and a moneyness of 95% (strike price of 221.25). Since it is a Put option and the strike price is lower than the current price, the option is out-of-the-money (OTM).

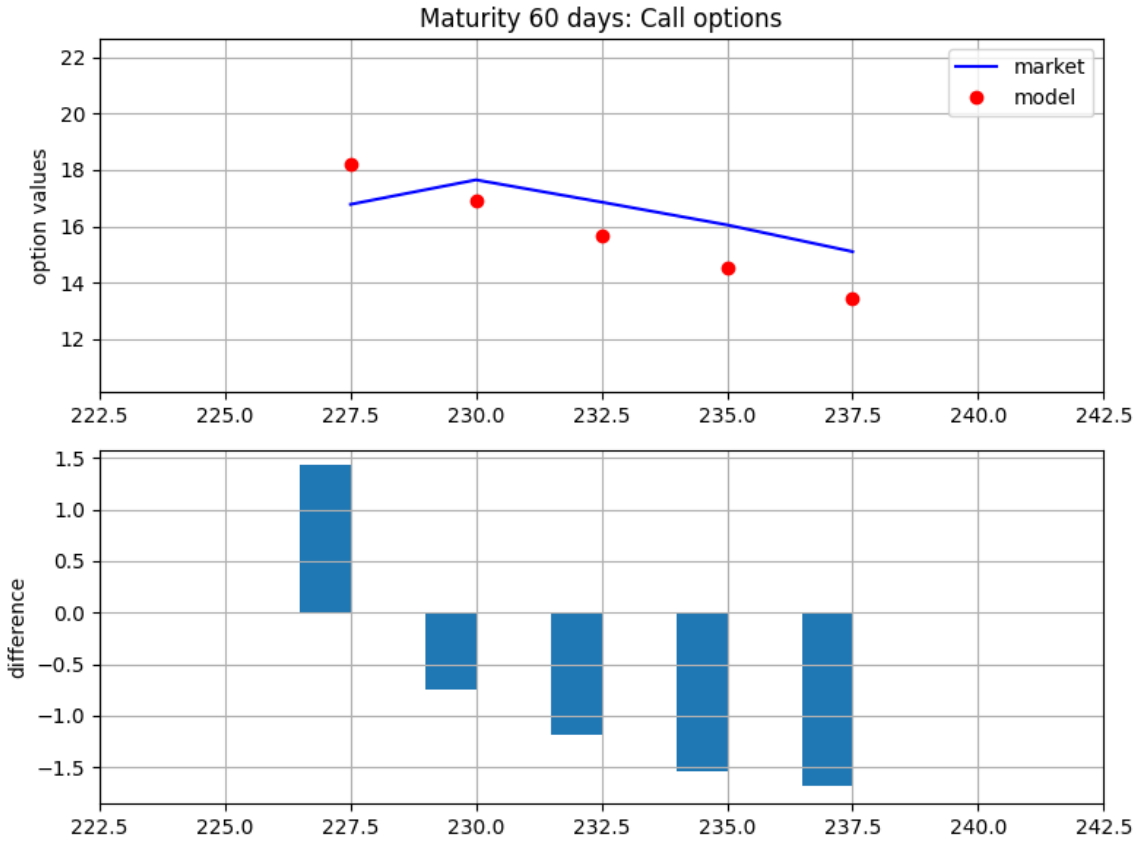


Figure 8: Bates calibrated model on Call prices

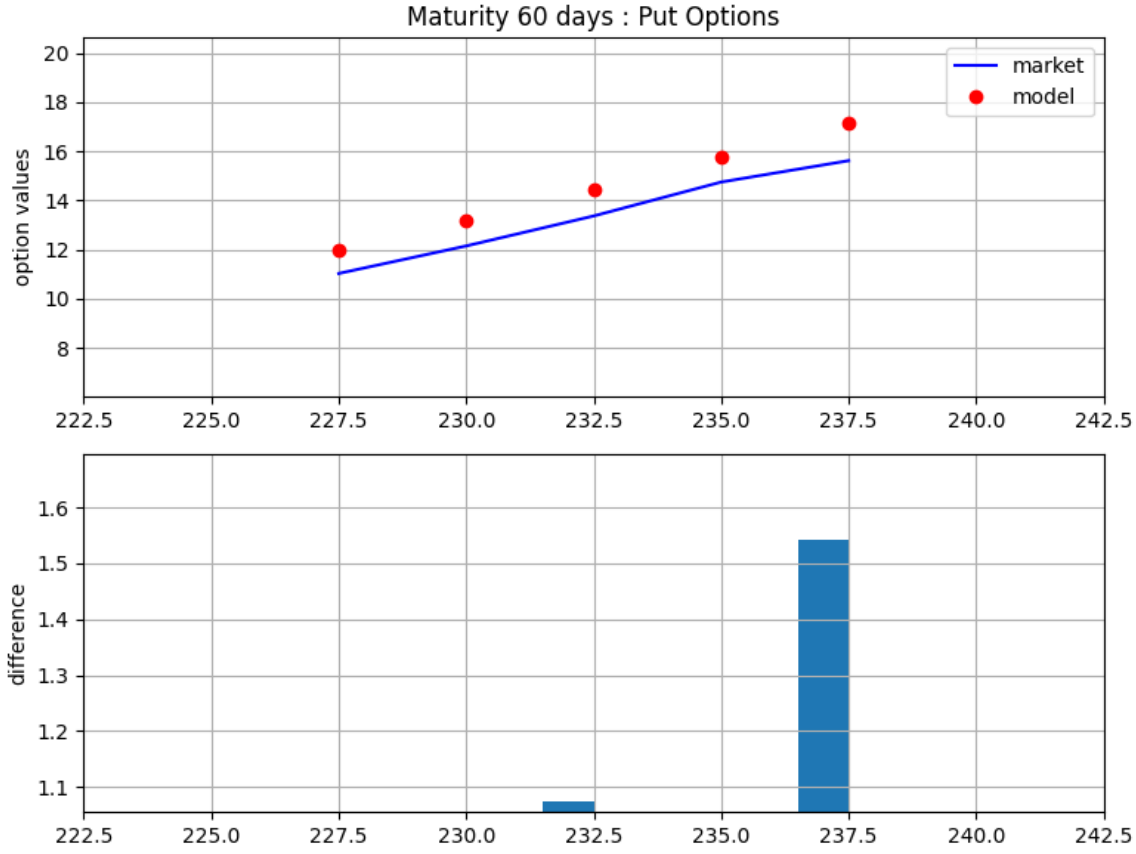


Figure 9: Bates calibrated model on put prices

Below are the input parameters required for valuing both call and put options using the Monte Carlo method. As the specific approach was not explicitly stated, we will use random path generation to determine option prices.

All parameters except volatility are provided in previous sections. Since the exact date of analysis is unknown, we cannot assess historical volatility based on market data. Therefore, as a proxy, we will use the implied volatility of the most similar option: an OTM Put with 60 days to maturity and a strike price of 227.5. Based on its price of 11.03, the implied volatility is calculated as follows:

- S0 = 232.90
- K = 227.5
- T = 60 / 250
- r = 0.02
- Market_price = 11.03

Based on the Black-Scholes model the implied volatility is 31.31
To assess the price of the Put option, we will use this implied volatility instead of relying on historical volatility.

Therefore based on the provided data and determined implied volatility and using the Monte-Carlo model the price of put option is 9.34$. and with 4% fees it's 9.72 $

# 3.A Calibrating CIR (1985) Model

As our client has very volatile demands, we must be ready to cater to all their needs when necessary so we should be ready with insights about the evolution of interest rates in the long term.

For that, we going to calibrate the CIR (1985)[11] model which is an enhanced version of Vasicek's (1977) model that includes a term to make the standard deviation of short rate changes proportional to $\sqrt{r_t}$. This yields the following SDE:

$$dr_t = k_r(\theta_r - r_t)dt + \sigma_r\sqrt{r_t}dz_t$$

Where the presence of $\sqrt{r_t}$ means that when the short rate increases, its standard deviation becomes higher.

The CIR model has also different function components:
In this model, prices of ZCBs paying 1 monetary unit at T take the following form:

$$B_0(T) = b_1(T)e^{-b_2(T)r_0}$$

where

$$b_1(T) = \left[\frac{2\gamma e^{((k_r+\gamma)T/2)}}{2\gamma + (k_r+\gamma)(e^{\gamma T}-1)}\right]^{\frac{2k_r\theta_r}{\sigma_r^2}}$$

$$b_2(T) = \frac{2(e^{\gamma T}-1)}{2\gamma + (k_r+\gamma)(e^{\gamma T}-1)}$$

$$\gamma = \sqrt{k_r^2 + 2\sigma_r^2}$$

As we have understood this model we are going to deal with model calibration in the following steps:

- Get market data
- Build up your valuation function according to your model
- Error function (difference between model output and observed market prices)
- Optimization function (minimizing error function)

### 0.0.27 Getting Bond Market Data

For Market Data we are going to consider Euribor rates as we operate mostly in a European setting.

The Rates are:

- 1 week = 0.648%
- 1 Month = 0.679%
- 3 months = 1.173%
- 6 months = 1.809%
- 12 months = 2.556%

We will divide the rates with 100 and maturities with 360 to express it into decimal(rates), fraction of years(maturities)

### 0.0.28 Bond Pricing and Forward Rates in CIR (1985) Model

When we perform our calibration, the goal is going to select model parameters ($\kappa_r$, $\theta_r$, $\sigma_r$, $r_0$) so to minimize the differences between the rates produced by the model, and the rates observed in practice. In other words, minimize $\Delta f(0,t)$:

$$\Delta f(0,t) \equiv f(0,t) - f^{CIR}(0,t;\kappa_r,\theta_r,\sigma_r,r_0)$$

where $f(0,t)$ is the current market implied forward rate between $0$ and time $t$. Formally, the forward rate from any time $t$ to $T$ is defined as:

$$f(t,T) \equiv -\frac{\partial B_t(T)}{\partial T}$$

Now, using the previous formula and the expressions we know from the CIR (1985) model, we can define the forward rate between times $t$ and $T$ and a set of parameters $\alpha$ in the following way:

$$f^{CIR}(t,T;\alpha) = \frac{\kappa_r\theta_r(e^{\gamma t}-1)}{2\gamma+(\kappa_r+\gamma)(e^{\gamma t}-1)} + r_0\frac{4\gamma^2 e^{\gamma t}}{(2\gamma+(\kappa_r+\gamma)(e^{\gamma t}-1))^2}$$

Of course, the fact we are considering forward rates makes a lot of sense; since we are going to be projecting things into the future, we would like to calibrate the model to the actual forward rates implied by the ZCB prices.

Now, when using forward rates between two different times (e.g., $t$ and $T$), our bond valuation equation would slightly change in notation. You will realize that there are no major changes in the formula but, for the sake of completeness, we include them next Svoboda (2002)[12].

$$B_t(T) = a(t,T)e^{-b(t,T)\mathbf{E_0^Q}(r_t)}$$

where

$$a(t,T) = \left[\frac{2\gamma e^{((k_r+\gamma)(T-t)/2)}}{2\gamma+(k_r+\gamma)(e^{\gamma(T-t)}-1)}\right]^{\frac{2k_r\theta_r}{\sigma_r^2}}$$

$$b(t,T) = \frac{2(e^{\gamma(T-t)}-1)}{2\gamma+(k_r+\gamma)(e^{\gamma(T-t)}-1)}$$

with

$$\mathbf{E_0^Q}(r_t) = \theta_r + e^{-\kappa_r t}(r_0-\theta_r)$$

and, of course, still with $\gamma = \sqrt{k_r^2+2\sigma_r^2}$.

Unfortunately, we do not typically get market quotes of the forward rates, which is what we need to properly execute our model calibration via minimizing $\Delta f(t,T)$ above for different $t$. Instead, we just have the yields (or rates) for different ZCBs and their maturities.

Luckily for us, there is a very simple way of going from bond yields to forward rates and vice versa (this, you should be familiar with even before the course). Suppose $Y(0,T)$ is the current bond yield (short rate) of a ZCB that pays 1 unit at maturity $T$:

$$f(0,T) = Y(0,T) + \frac{\partial Y(0,T)}{\partial T}T$$

Simultaneously, we know that the price of that bond today, given yield $Y(0,T)$ should solve the following equation:

$$B_T(T) = B_0(T)e^{Y(0,T)T} \Leftrightarrow Y(0,T) = \frac{logB_T(T) - logB_0(T)}{T}$$

and since we have normalized the face value of the bond at maturity to 1:

$$Y(0,T) = -\frac{logB_0(T)}{T}$$

Similarly, we can derive capitalization factors (the continuous yield of a unit ZCB) and equivalent annualized continuous rates. For example, capitalization factor, $f_s^{3m}$, for the 3 months Euribor rate, $Eur_{3m}$ would be:

$$f_s^{3m} = 1 + 90/360 \times Eur_{3m}$$

The equivalent annualized continuous rate, $f_c^{3m}$, is therefore defined as:

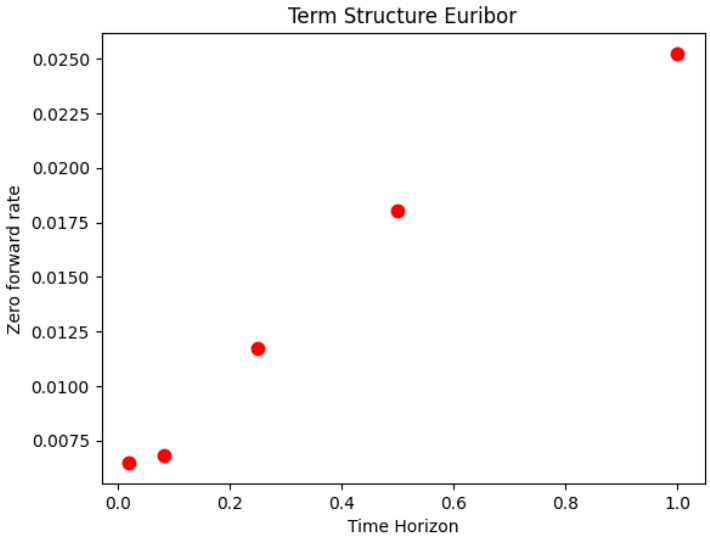


Figure 10: Euribor Term Structure

$$f_c^{3m} = 360/90 \times log(f_s^{3m})$$

which ensures the following relationship is satisfied:

$$1 + 90/360 \times f_s^{3m} = e^{90/360 f_c^{3m}}$$

So, now that we know all this, let's define the current short-term rate ($r_0$), the capitalization factors and the zero-forward rates implied by the Euribor rates observed in the market in Figure 10.

### 0.0.29 Interpolation of Market Rates

Unfortunately, as we observed in the previous graph, there are a limited amount of rates quoted in the market. However, in order to calibrate the parameters of our model, we would like to have as many inputs (data points) as possible.

In order to do that, there is actually one common solution: interpolate the term structure of forward rates. This basically consists of fitting a function that can replicate the observed term structure and infer what would be the forward rate of a given maturity for which there is no market quote.

There are several ways to interpolate a term structure (e.g., linear interpolation, fitting a model such as Nelson-Siegel using OLS, etc). Here, we will use a common way of interpolation, which is cubic spline interpolation.

Now, we are going to take advantage of the scipy package in Python and its built-in spline functions splrep and splev . [14]

We created 52 equally spaced maturities between 0 and 1 because we need to interpolate weekly rates for period of 12 months, we can see the same in Figure 11:

As you can see, we do not perfectly match the observed market data, but we are pretty close. In turn, we have many more data points that we can use to calibrate our interest rate model to observed (or, better said, interpolated from observed) market quotes.

### 0.0.30 CIR forward Rates

Finally, before jumping to the pure calibration, we wrote one last function. We will calibrate forward rates of the CIR (1985) model. Hence, we defined the forward rate expression in the CIR model, $f^{CIR}(t, T; \alpha)$. Remember that we have already seen this expression before:

$$f^{CIR}(t, T; \alpha) = \frac{\kappa_r \theta_r (e^{\gamma t} - 1)}{2\gamma + (\kappa_r + \gamma)(e^{\gamma t} - 1)} + r_0 \frac{4\gamma^2 e^{\gamma t}}{(2\gamma + (\kappa_r + \gamma)(e^{\gamma t} - 1))^2}$$

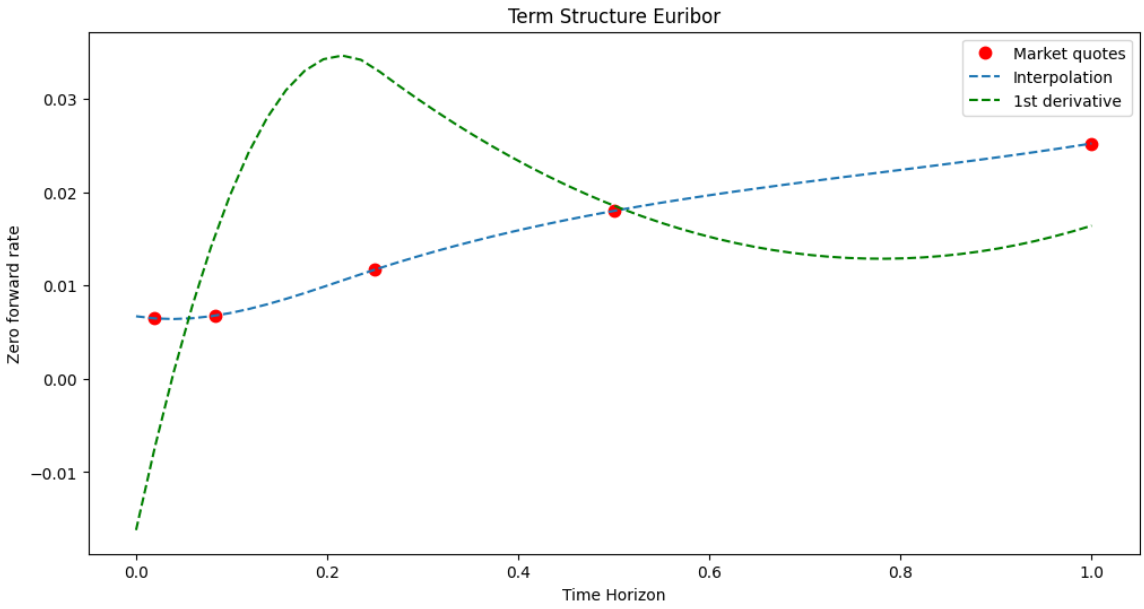


Figure 11: Interpolated Euribor Term Structure

**Algorithm 5** CIR Forward Rate

**procedure** CIR_FORWARD_RATE(alpha)
  **Input:**
    $alpha$: A set of parameters including $\kappa_r$, $\theta_r$, and $\sigma_r$
  Extract parameters from $alpha$:
    $\kappa_r, \theta_r, \sigma_r \leftarrow \alpha$
  Define $t \leftarrow$ mat_list_n (maturity list)
  Calculate $g \leftarrow \sqrt{\kappa_r^2 + 2\sigma_r^2}$
  **Step 1: Calculate** $s1$
  $s1 \leftarrow \frac{\kappa_r \theta_r (\exp(g \cdot t) - 1)}{2g + (\kappa_r + g)(\exp(g \cdot t) - 1)}$
  **Step 2: Calculate** $s2$
  $s2 \leftarrow r0 \left( \frac{4g^2 \cdot \exp(g \cdot t)}{2g + (\kappa_r + g)(\exp(g \cdot t))^2} \right)$
  **Step 3: Calculate Forward Rate**
  **return** $s1 + s2$
**end procedure**

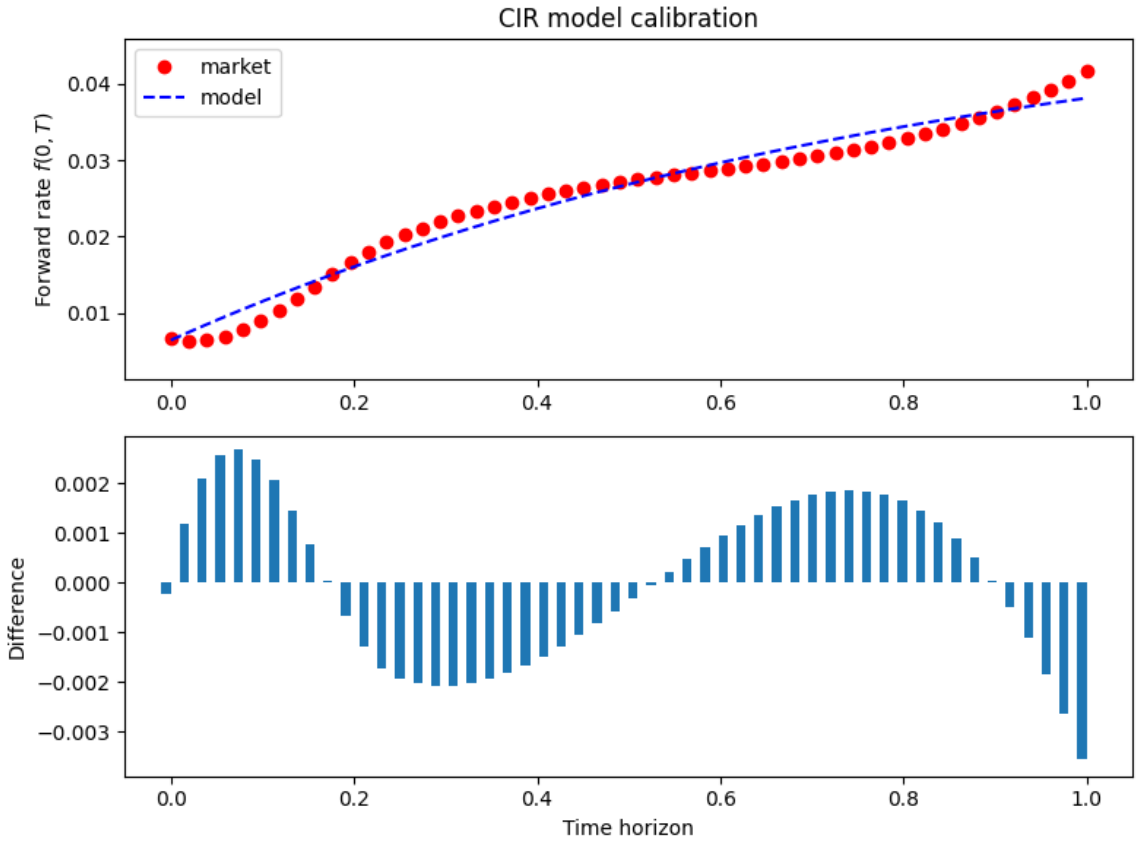


Figure 12: CIR model Calibration

### 0.0.31 Error Function

Next, we will define our error function, which basically is equivalent to other error functions we have defined before. In this case, we are using Mean Squared Error (MSE). This is, given $\alpha$ equal to the set of parameters to calibrate:

$$min \frac{1}{M} \sum_{m=0}^{M} \left( f(0, m\Delta t) - f^{CIR}(0, m\Delta t; \alpha) \right)^2$$

with $M = T/\Delta t$, that is, the number of market data points between $0$ and $T$.

### 0.0.32 Optimization

Finally, we just need to create our optimization function to minimize the difference between model and market rates, very much in the same spirit as we have done a few times before for other models:

### 0.0.33 Results

Now, we are ready to perform the whole calibration process as usual. First, let's run our calibration function to obtain the parameters:

we obtain parameters that closely match market data:

$$\kappa_\nu = 0.998$$
$$\theta_\nu = 0.107$$
$$\sigma_\nu = 0.001$$

Then, let's see graphically the results of our calibration given the previous parameters in Figure 12:

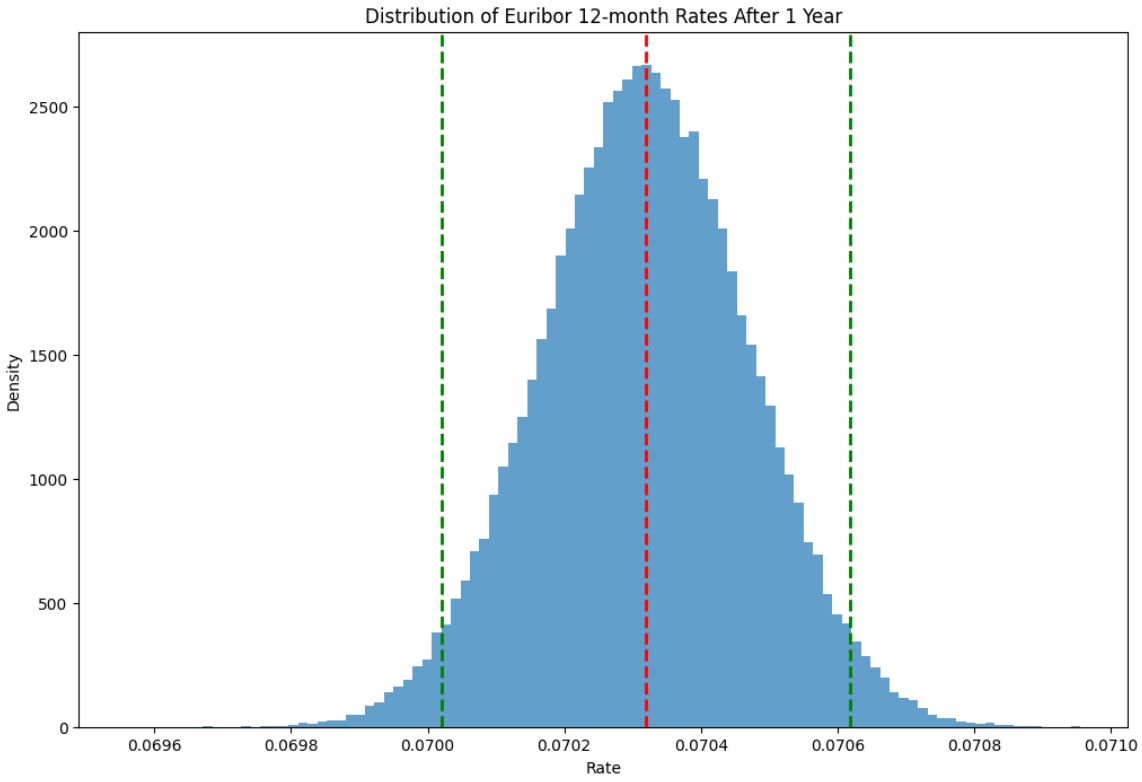


Figure 13: Simulated 12-month Euribor rates histogram

## 3.B Using CIR for Euribor daily rates simulations via Monte-Carlo

We'll simulate the Euribor 12-month rates using the Cox-Ingersoll-Ross (CIR) model with the pre-calibrated parameters from the previous section stated below. Euribor, which stands for Euro Interbank Offered Rate, is a key benchmark interest rate in the European money market. It represents the average interest rate at which a large panel of European banks lend unsecured funds to one another in the euro wholesale money market for a given period.

$$\kappa_\nu = 0.998$$
$$\theta_\nu = 0.107$$
$$\sigma_\nu = 0.001$$

### i. Range of 12-month Euribor in the next year

Using a 95% confidence level, we find that the range for the 12-month Euribor rate in the next year is [0.0700, 0.0706]. This means that we can be 95% confident that the Euribor 12-month rate will fall within this range after one year, based on our simulations. Figure 13 represents a distribution histogram with green lines outlining the confidence interval border values, and the red line outlining simulations' mean value.

### ii. Expected value of 12-month Euribor in 1 year

The expected value of the 12-month Euribor rate, based on our simulations, is 0.0703 or 7.03%. The simulated value is significantly higher than the 5% that we used for option pricing exercises earlier. That means, that a price recalculation with the new simulated value would be a good idea to stick closer to actual market behavior.

### iii. Impact on product pricing

To assess the impact on product pricing, we need to compare the expected future rate of 7.03% with the current 12-month Euribor rate. If we return to pricing of a Asian put for earlier exercie, the expected increase in the 12-month Euribor rate would likely lead to increase of the option price from \$4.53 to \$4.78 or \$4.71 to \$4.98 with 4% fee applied.

# Bibliography


[1] Heston, S. L. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The Review of Financial Studies, 6(2), 327-343.

[2] Wilmott, P. (2006). Paul Wilmott on Quantitative Finance. John Wiley & Sons.

[3] Kloeden, P. E., & Platen, E. (1992). Numerical Solution of Stochastic Differential Equations. Springer-Verlag.

[4] Robert C. Merton, Option pricing when underlying stock returns are discontinuous, Journal of Financial Economics, Volume 3, Issues 1–2, 1976, Pages 125-144, ISSN 0304-405X

[5] Gatheral, Jim. *The Volatility Surface: A Practitioner's Guide*. John Wiley & Sons Inc., 2006.

[6] Hilpisch, Yves. *Derivatives Analytics with Python: Data Analysis, Models,Simulation, Calibration and Hedging.* John Wiley & Sons, 2015.

[7] Carr, Peter & Stanley, Morgan & Madan, Dilip. (2001). Option Valuation Using the Fast Fourier Transform. J. Comput. Finance. 2. 10.21314/JCF.1999.043.

[8] Bates, David S. 'Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options'. The Review of Financial Studies 9.1 (1996): 69-107.

[9] Cherubini, Umberto, et al. Fourier transform methods in finance. John Wiley & Sons, 2010

[10] Galluccio, Stefano, and Yann Lecam. "Implied calibration and moments asymptotics in stochastic volatility jump diffusion models." Available at SSRN 831784 (2008).

[11] Cox, John C., et al. "An Intertemporal General Equilibrium Model of Asset Prices." *Econometrica: Journal of the Econometric Society*, 1985, pp. 363-384.

[12] Svoboda, Simona. *An Investigation of Various Interest Rate Models and Their Calibration in the South African Market.* 2002. University of the Witwatersrand, Dissertation. https://www.econbiz.de/Record/interest-rate-model-theory-with-reference-to-the-south-african-market-van-wijck-tjaart/10009442156

[13] Lewis, Alan L. "A simple option formula for general jump-diffusion and other exponential Lévy processes." Available at SSRN 282110 (2001).

[14] Scipy : (https://docs.scipy.org/doc/scipy/)