Calibration of SABR Model Using Swaption Volatility in Python

Ali Sidibe*
bromasidibe@gmail.com

November 25, 2025

1 Introduction

In today's financial markets, understanding how implied volatility behaves is essential for valuing and managing the risks of derivative products. Among the different models developed for this purpose, the SABR model (Stochastic Alpha, Beta, Rho) has become a reference point, especially in interest rate derivatives. Introduced by Hagan, Kumar, Lesniewski, and Woodward, it provides a flexible and intuitive way to reproduce the volatility smiles that traders actually see on the screens.

Compared with more traditional frameworks—like Black's model with constant volatility, or even local-volatility models such as CEV—SABR adds a crucial ingredient: it makes volatility itself stochastic. This seemingly simple idea allows the model to capture important market features, such as skews and curvature in the implied volatility surface, while still remaining reasonably tractable.

One thing to keep in mind is that SABR does not directly produce option prices. What it gives you is an approximation of the implied volatility. That volatility is then plugged into Black's formula to get the price of a swaption, a cap, a floor, and so on. This two-step structure is one of the reasons practitioners like the model so much: it stays easy to use while fitting real-world market behaviour remarkably well.

2 Dynamics

The SABR model of Hagan et al. [**?**] is defined by the stochastic differential equations

$$\begin{aligned} df_t &= \alpha_t \, f_t^{\beta} \, dW_t^1, \\ d\alpha_t &= \nu \, \alpha_t \, dW_t^2, \\ \mathbb{E}\big[dW_t^1 \, dW_t^2\big] &= \rho \, dt, \end{aligned} \tag{1}$$

with initial values f_0 and α_0.

In this formulation:

- f_t is the forward rate,

*University Pantheon Sorbonne

- α_t is the stochastic volatility,
- W_t^1 and W_t^2 are Brownian motions with instantaneous correlation ρ.

The model is parameterised by:

- **Initial volatility α_0.** Controls the overall *level* of the implied volatility surface. Increasing α_0 shifts the entire smile upward without significantly changing its shape.
- **Elasticity parameter β.** Determines how strongly volatility depends on the forward level f_t.
  - High β (close to 1): lognormal-like behaviour; the smile is flatter for high strikes and steeper for low strikes.
  - Low β (close to 0): normal-like behaviour; the smile becomes more symmetric.

  The parameter β therefore controls the *level of skewness* implied by the forward-rate dynamics.
- **Vol-vol ν.** Controls the amount of stochasticity in the volatility process. Larger ν amplifies the curvature of the smile, producing a more pronounced *smile or smirk*. With $\nu = 0$, the model reduces to a deterministic-volatility model with little curvature.
- **Correlation ρ.** Dictates the direction and steepness of the skew.
  - $\rho < 0$: produces a strong negative skew, typical in interest-rate and FX markets.
  - $\rho > 0$: produces a positive skew.

  The magnitude of ρ controls how steep the skew is.

3 Market data

For this SABR calibration, we take the swaption market volatilities corresponding to the option whose strike equals the forward swap rate (ATM swaption volatility). Specifically:

- Expiry: the time until the option starts (e.g., 5 years).
- Tenor: the length of the underlying swap (e.g., 1 year).
- ATM strike: the strike equal to the forward swap rate for the given expiry and tenor.
- ATM volatility: the market implied volatility for this swaption.

Once the ATM volatility is obtained for each expiry/tenor pair, it is used to compute the SABR parameter α. The remaining parameters ρ and ν are then calibrated to reproduce the full market volatility smile across different strikes.

	1Y1Y	1Y2Y	1Y3Y	1Y5Y	1Y10Y	5Y1Y	5Y2Y	5Y3Y	5Y5Y	5Y10Y	10Y1Y	10Y2Y	10Y3Y	10Y5Y	10Y10Y
-200	91.570	83.270	73.920	55.190	41.180	67.800	57.880	53.430	41.990	34.417	55.160	51.170	48.220	40.550	33.601
-150	62.030	61.240	56.870	44.640	35.040	49.090	46.410	44.440	36.524	30.948	44.320	42.900	41.430	35.891	30.509
-100	44.130	46.570	44.770	36.510	30.207	38.400	39.033	38.180	32.326	28.148	37.368	37.078	36.400	32.181	27.978
-50	31.224	35.807	35.745	30.242	26.619	31.485	33.653	33.437	29.005	25.954	32.259	32.622	32.439	29.144	25.926
-25	26.182	31.712	32.317	27.851	25.351	29.060	31.531	31.536	27.677	25.136	30.210	30.800	30.796	27.857	25.086
0	22.500	28.720	29.780	26.070	24.470	27.260	29.830	29.980	26.600	24.510	28.540	29.280	29.400	26.740	24.370
25	20.960	27.120	28.290	24.980	23.980	26.040	28.560	28.760	25.730	23.990	27.310	28.090	28.270	25.800	23.760
50	21.400	26.840	27.800	24.560	23.820	25.320	27.650	27.820	25.020	23.560	26.450	27.200	27.380	25.020	23.240
100	24.340	28.510	28.770	25.120	24.250	24.940	26.710	26.670	24.060	22.910	25.610	26.120	26.180	23.870	22.440
150	27.488	31.025	30.725	26.536	25.204	25.320	26.540	26.200	23.570	22.490	25.520	25.720	25.580	23.170	21.900
200	30.297	33.523	32.833	28.165	26.355	25.980	26.760	26.150	23.400	22.250	25.780	25.710	25.370	22.800	21.560

Figure 1: Market volatility of swaption by expiry and moneyness

4 Calibration

In the SABR model, four parameters must be calibrated: β, α, ρ, and ν. Calibrating all four simultaneously is generally not recommended.

A more stable calibration procedure is:

1. **Estimate** β first.
2. **Estimate** α_0**,** ρ**, and** ν **directly**; this approach is most appropriate for **FX and equity options**.
3. **Estimate** ρ **and** ν **directly, and infer** α_0 from ρ, ν, and the at-the-money volatility σ_{ATM}. This method is particularly well suited for **interest rate derivatives**.

Hagan, Kumar, Lesniewski, and Woodward observe that the market smile can be fitted reasonably well for almost any fixed value of β. Attempting to calibrate β to market data is discouraged because it tends to fit market noise rather than meaningful structure. They therefore recommend extracting β from historical observations of the *backbone*, the curve traced out by the at-the-money volatility σ_{ATM}.

In practice, β is estimated first, and its precise value is not critical, since different choices of β do not significantly change the overall shape of the implied volatility smile. Once β has been fixed, the remaining parameters can be calibrated using either of the two approaches described above, depending on the asset class.

4.1 Estimating βbeta

From the SABR implied volatility formula, the at-the-money volatility σ_{ATM} is obtained by setting $f = K$ in the expression for the SABR implied volatility. This yields

$$\sigma_{\text{ATM}} = \sigma_B(f, f) = \alpha_0 f^{1-\beta}\left[1 + \left(\frac{1-\beta}{2}\right)\left(\frac{\alpha_0^2}{24 f^{2-2\beta}} + \frac{\rho\beta\nu\alpha_0}{4 f^{1-\beta}} + \frac{2-3\rho^2}{24}\nu^2\right)T\right]. \tag{2}$$

Taking logarithms gives the approximate relationship

$$\ln \sigma_{\text{ATM}} \approx \ln \alpha_0 \; - \; (1-\beta)\, \ln f. \tag{3}$$

Thus, β may be estimated by performing a linear regression on a time series of $\ln \sigma_{\text{ATM}}$ versus $\ln f$. Alternatively, β can simply be chosen based on prior beliefs about whether a stochastic normal model ($\beta = 0$), a stochastic lognormal model ($\beta = 1$), or a CIR-type model ($\beta = \frac{1}{2}$) is appropriate.

In practice, the exact choice of β has limited influence on the overall shape of the implied volatility smile generated by the SABR model, and is therefore not critical. However, the value of β can affect the sensitivities (the Greeks). Bartlett [**?**] provides refined Greeks and shows that they are less sensitive to the choice of β.

```
spot_rate = {0.5: 0.0025, 1: 0.0030, 2: 0.0033, 3: 0.0034,
             4: 0.0035,5: 0.0036, 7: 0.0040, 10: 0.0045,
             15: 0.0050, 20: 0.0053, 30: 0.0055}
tenors = np.array(list(discount_factors.keys()))
rates = np.array(list(discount_factors.values()))
spot_rate_interp = PchipInterpolator(tenors,rates)

def forward_rate(start_date, end_date):
    DF1 = math.exp(-float(spot_rate_interp(start_date)) * start_date)
    DF2 = math.exp(-float(spot_rate_interp(end_date)) * end_date)
    return -math.log(DF2 / DF1) / (end_date - start_date)

vol_atm = (df_market_vol.T)[0].values/100
texp =  (df_market_vol.T).index
t_exp = [(int(x.split('Y')[0]), int(x.split('Y')[1])) for x in texp]
forwar_rate = [forward_rate(t, t+expiry )for (t,expiry) in t_exp]
Y = np.log(vol_atm)
X = np.log(forwar_rate).reshape(-1, 1)
reg = LinearRegression().fit(X, Y)
beta = reg.coef_[0] + 1
print(f'beta = {beta}')
```

5 Full calibration Estimating α, ρ, and ν

Once $\hat{\beta}$ is set, the remaining task is to estimate the parameters α, ρ, and ν. This can be achieved by minimizing the errors between the model volatilities and the observed market volatilities σ_i^{mkt} (typically obtained from interest-rate derivatives) with identical maturity T.

A common approach is to use a least-squares objective function such as the sum of squared errors (SSE):

$$(\hat{\alpha}, \hat{\rho}, \hat{\nu}) = \arg\min_{\alpha,\rho,\nu} \sum_i \Big(\sigma_i^{\text{mkt}} - \sigma_B(f_i, K_i; \alpha, \rho, \nu)\Big)^2. \tag{4}$$

After estimating α, β, ρ, and ν, we substitute these parameters into the SABR implied volatility expression in equation (3) to obtain σ_B. The resulting volatility is then used in the Black formula (equation (2)) to compute option prices.

Other choices of objective functions are possible. For example, West [**?**] proposes a weighted least-squares approach in which the weights are proportional to vega.

```
    def get_sabr_volatility(F, K, T, alpha, rho, nu, beta=0.9):
    if abs(F - K) < 1e-12:
        FK_beta = F ** (1 - beta)
        term1 = ((1 - beta) ** 2 / 24) * (alpha ** 2) / (F ** (2 - 2 * beta))
        term2 = 0.25 * rho * beta * nu * alpha / (F ** (1 - beta))
        term3 = ((2 - 3 * rho ** 2) / 24) * nu ** 2
        return alpha / FK_beta * (1 + (term1 + term2 + term3) * T)
    FK_beta = (F * K) ** ((1 - beta) / 2)
    logFK = np.log(F / K)
    z = (nu / alpha) * FK_beta * logFK
    x_z = np.log((np.sqrt(1 - 2 * rho * z + z ** 2) + z - rho) / (1 - rho))
    term1 = ((1 - beta) ** 2 / 24) * (alpha ** 2) / (FK_beta ** 2)
    term2 = 0.25 * rho * beta * nu * alpha / FK_beta
    term3 = ((2 - 3 * rho ** 2) / 24) * nu ** 2
    numerator = alpha * (1 + (term1 + term2 + term3) * T) * z
    denominator = FK_beta * (1 + ((1 - beta) ** 2 / 24) * logFK ** 2 + ((1 - beta) **
    sigma = numerator / denominator
    return sigma

def calibrate_sabr(expiry, tenor, forward_swap_rate, df_market_vol):
    key = f"{tenor}Y{expiry}Y"
    vol_market = df_market_vol[key].values / UNIT_PCT
    moneyness_change = df_market_vol.index.values / UNIT_BPS
    def residuals(params):
        alpha, rho, nu = params
        error = 0
        for i, vol in enumerate(vol_market):
            K = forward_swap_rate + moneyness_change[i]
            vol_sabr = get_sabr_volatility(forward_swap_rate,  K, expiry, alpha, rho,
            error = error + (vol_sabr - vol)**2
        return error
    initial_guess = [0.02, 0.0, 0.5]
    bounds = ([0.0001, -0.999, 0.0001], [1.0, 0.999, 2.0])
    res = least_squares(residuals, initial_guess, bounds=bounds, max_nfev=6000)
    return  res.x

expiry = 1
tenor = 5
forwar_swap_rate = 0.0352554715328091
sabr_calibration_params = calibrate_sabr(expiry, tenor, forwar_swap_rate, df_market_vo
print(f'alpha =  {sabr_calibration_params[0]} rho =  {sabr_calibration_params[1]} vol
#alpha =  0.18546 rho =  -0.5645 vol_of_vol =  1.3042
```

6 Second Parameterization — Estimating ρ and ν

We can reduce the number of parameters to be estimated by using the ATM volatility σ_{ATM} to obtain α via equation (4), rather than estimating α directly. This means that

we only need to estimate ρ and ν, and obtain α by inverting equation (4). In particular, α is the root of the cubic equation

$$\left(\frac{(1-\beta)^2 T}{24 f^{2-2\beta}}\right)\alpha^3 + \left(\frac{\rho\beta\nu T}{4 f^{1-\beta}}\right)\alpha^2 + \left(1 + \frac{2-3\rho^2}{24}\nu^2 T\right)\alpha - \sigma_{\text{ATM}} f^{1-\beta} = 0. \tag{6}$$

West [**?**] notes that this cubic equation may have more than one real root, and recommends selecting the smallest positive root when this occurs.

It is relatively straightforward to estimate the parameters using this second parameterization. In our minimization algorithm, at every iteration we compute $\alpha = \alpha(\rho, \nu)$ by solving equation (6). For example, the sum of squared errors from equation (5) becomes

$$(\hat{\rho}, \hat{\nu}) = \arg\min_{\rho,\nu} \sum_i \left(\sigma_i^{\text{mkt}} - \sigma^B(f_i, K_i, \alpha(\rho,\nu), \rho, \nu)\right)^2. \tag{7}$$

This optimization takes longer to converge because at every iteration step the algorithm proposes new values for ρ and ν, but must then use a root-finding routine to solve equation (6) for $\alpha(\rho, \nu)$, using the inputs β, ρ, ν, along with f, K, σ_{ATM}, and T.

The resulting values of ρ, ν, and $\alpha(\rho, \nu)$ are plugged into equation (3) to produce σ^B, which is used in evaluating the objective function (7). The objective value is then compared against the tolerance (or another convergence criterion) and the algorithm proceeds to the next iteration.

```
def solve_alpha_from_atm_vol(beta, rho, nu, sigma_ATM, f, T):
    A = (1 - beta**2) / (24 * f**(2*(1-beta)))
    B = (rho * beta * nu) / (4 * f**(1-beta))
    C = 1
    D = -sigma_ATM * f**(1-beta)

    roots = np.roots([A, B, C, D])

    pos = [r.real for r in roots if abs(r.imag) < 1e-10 and r.real > 0]
    return min(pos)


def calibrate_sabr_partial(expiry, tenor, forward_swap_rate, vol_market,moneyness_char
    key = f"{tenor}Y{expiry}Y"
    vol_market = df_market_vol[key].values / UNIT_PCT
    moneyness_change = df_market_vol.index.values / UNIT_BPS
    sigma_atm = vol_market[moneyness_change == 0][0]

    def residuals(params):
        rho, nu = params
        error = 0
        alpha = solve_alpha_from_atm_vol(beta,rho,nu,vol_atm,forward_swap_rate,expiry)
        for i, vol in enumerate(vol_market):
            K = forward_swap_rate + moneyness_change[i]
            vol_sabr = get_sabr_volatility(forward_swap_rate,  K, expiry, alpha, rho,
            error = error + (vol_sabr - vol)**2
```

```
        return error
    initial_guess = [0.0, 1]
    bounds = ([-0.999, 0.0001], [0.999, 2.0])
    res = least_squares(residuals, initial_guess, bounds=bounds, max_nfev=6000)
    alpha = solve_alpha_from_atm_vol(beta,res.x[0],res.x[1],vol_atm,forward_swap_rate
    print(alpha)
    return  (alpha,res.x)

##Test
expiry = 1
tenor = 5
forwar_swap_rate = 0.0352554715328091
key = f"{tenor}Y{expiry}Y"
vol_market = df_market_vol[key].values / UNIT_PCT
moneyness_change = df_market_vol.index.values / UNIT_BPS
vol_atm = vol_market[moneyness_change == 0][0]
sabr_calibration_params = calibrate_sabr2(expiry, tenor, forwar_swap_rate, vol_market
rho = sabr_calibration_params[0]
vol_of_vol = sabr_calibration_params[1]
alpha = solve_alpha_from_atm_vol(beta,rho,vol_of_vol,vol_atm,forwar_swap_rate,expiry)
print(f'alpha =  {alpha} rho =  {rho} vol_of_vol =  {vol_of_vol}')
#alpha =  0.3832 rho =  -0.975 vol_of_vol =  1.99
```