The SABR Model for Stochastic Volatility

Gradev, Aleksandar Grönum, Wald Milosavljevic, Aleksandar Berger, Lennart

17.01.2026

This report implements and analyzes the SABR (Stochastic Alpha, Beta, Rho) model for option pricing with stochastic volatility, as introduced by Hagan, Kumar, Lesniewski, and Woodward [2002]. We use numerical simulation methods following Glasserman [2004] and stochastic calculus theory from Shreve [2004].

1 Stochastic Differential Equations

The Black-Scholes model assumes that the price $S(t)$ of an asset follows the following stochastic differential equation:

$$dS(t) = S(t)\left(\mu\, dt + \sigma\, dW(t)\right). \tag{1}$$

The first part, $dS(t) = S(t)\mu\, dt$, can also be written as

$$S'(t) = \frac{dS(t)}{dt} = \mu S(t). \tag{2}$$

It is an ordinary differential equation that describes the change $S'(t)$ as a function of $S(t)$. The solution of such a differential equation are functions $S(t)$ where (2) holds for all $t \geq 0$. When an initial value S_0 is given (that is, we assume that $S(0) = S_0$), then this solution is uniquely defined. In our case a simple computation shows that

$$S(t) = S_0 e^{\mu t} \tag{3}$$

solves (2). This can be easily verified by substituting $S(t)$ and its derivative $S'(t)$ into (2).

The second part of Eq. (1) contains the term $dW(t)$. Here $W(t)$ denotes the so-called *Wiener process* (a standardized Brownian motion) [Shreve, 2004]. It is a continuous-time stochastic process (i.e., a random path) with the properties:

1. $W(0) = 0$, almost surely.
2. W has independent increments: for every $s, t > 0$, the future increments $W(t+s) - W(t)$ are independent of $W(t)$.
3. $W(t+s) - W(t)$ is normally distributed with mean 0 and variance s,
$$W(t+s) - W(t) \sim \mathcal{N}(0, s).$$
4. $W(t)$ is continuous in t with probability 1.

By Itô's lemma [Shreve, 2004], the solution to the stochastic differential equation (1) yields the geometric Brownian motion:

$$\ln\left(\frac{S(t)}{S_0}\right) = \left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W(t). \tag{4}$$

For numerical simulation, we use the *Euler-Maruyama scheme* [Glasserman, 2004] to obtain an approximate realization of a solution path. For a time step δt and standard normal random variable $Z \sim \mathcal{N}(0,1)$:

$$S(t+\delta t) = S(t) + S(t)\left(\mu \delta t + \sigma\sqrt{\delta t}\, Z\right). \tag{5}$$

Alternatively, using the exact solution for geometric Brownian motion:

$$S(t+\delta t) = S(t) \exp\left[\left(\mu - \frac{\sigma^2}{2}\right)\delta t + \sigma\sqrt{\delta t}\, Z\right]. \tag{6}$$

However, the Black-Scholes model has a critical limitation: it assumes *constant volatility* σ. This is inconsistent with observed market phenomena such as the *volatility smile* and *volatility clustering*. This motivates the development of stochastic volatility models, such as the SABR model discussed in the next section.

2 The SABR Model

The SABR model[1] (Stochastic Alpha, Beta, Rho) was introduced by Hagan et al. [2002] to address the limitations of constant volatility models. The SABR model assumes that volatility itself is a stochastic process. It is widely used in interest rate and foreign exchange derivatives markets to capture the volatility smile observed in option prices.

The SABR model describes the evolution of a forward price $F(t)$ and its stochastic volatility $\alpha(t)$ through the following coupled stochastic differential equations:

$$dF(t) = \alpha(t)F(t)^{\beta}\, dW_1(t), \tag{7}$$
$$d\alpha(t) = \nu\, \alpha(t)\, dW_2(t), \tag{8}$$

where $W_1(t)$ and $W_2(t)$ are Wiener processes with correlation ρ:

$$\mathbb{E}[dW_1(t)\, dW_2(t)] = \rho\, dt. \tag{9}$$

The model has five parameters:

- F_0, the initial forward price. For a spot price S_0 and risk-free rate r, we have $F_0 = S_0 e^{rT}$ where T is the expiration time.
- α_0, the initial volatility level.
- $\beta \in [0, 1]$, the *elasticity parameter* that controls how volatility scales with the asset price. Common choices are:
  - $\beta = 0$: normal volatility model (absolute volatility independent of price level),
  - $\beta = 0.5$: CEV (Constant Elasticity of Variance) model,
  - $\beta = 1$: lognormal volatility model.
- $\nu \geq 0$, the *volatility of volatility* (vol-of-vol). Higher ν leads to more volatile $\alpha(t)$.
- $\rho \in [-1, 1]$, the correlation between price movements and volatility changes. Typically $\rho < 0$ in equity markets, reflecting the so-called *leverage effect*: when prices decline, volatility tends to increase.

Notice that the SABR model reduces to the Black-Scholes model if $\beta = 1$ and $\nu = 0$. In this case, $\alpha(t) = \alpha_0$ is constant, and equation (7) becomes

$$dF(t) = \alpha_0 F(t)\, dW_1(t), \tag{10}$$

which is equivalent to the geometric Brownian motion in equation (1) with $\mu = 0$ (drift-free under the forward measure) and $\sigma = \alpha_0$.

For simplicity, in this project we will work with the *spot price* $S(t)$ instead of the forward price. We introduce a drift parameter μ in equation (7) to obtain:

$$dS(t) = \mu S(t)\, dt + \alpha(t)S(t)^{\beta}\, dW_1(t). \tag{11}$$

The volatility process $\alpha(t)$ follows a lognormal process without drift. Applying Itô's lemma to $\ln(\alpha(t))$ yields:

$$\ln(\alpha(t)) = \ln(\alpha_0) - \frac{\nu^2}{2}t + \nu W_2(t). \tag{12}$$

This ensures that $\alpha(t) > 0$ almost surely for all $t > 0$, which is crucial for the model to remain well-defined.

[1] We use the notation from Hagan et al. [2002].

3 Numerical Simulation of SABR Paths

Since the SABR model does not have a closed-form solution for the price distribution when $\beta \neq 1$ and $\nu > 0$, we must rely on numerical simulation [Glasserman, 2004]. We use the *Euler-Maruyama scheme* to discretize the stochastic differential equations.

3.1 Euler-Maruyama Discretization

For a time horizon T divided into n steps with step size $\delta t = T/n$, we discretize the SABR equations as follows.

Volatility Process: The volatility $\alpha(t)$ follows a lognormal process. Using the exact solution from equation (12), we obtain:

$$\alpha(t+\delta t) = \alpha(t) \exp\left[-\frac{\nu^2}{2}\delta t + \nu\sqrt{\delta t}\, Z_2\right], \tag{13}$$

where $Z_2 \sim \mathcal{N}(0,1)$ is a standard normal random variable. This formulation guarantees that $\alpha(t) > 0$ for all time steps.

Price Process: For the price process $S(t)$, we distinguish two cases:

1. **Case $\beta = 1$:** We use the exact (log-Euler) update to preserve positivity:

$$S(t+\delta t) = S(t) \exp\left[\left(\mu - \frac{\alpha(t)^2}{2}\right)\delta t + \alpha(t)\sqrt{\delta t}\, Z_1\right], \tag{14}$$

   where $Z_1 \sim \mathcal{N}(0,1)$ is correlated with Z_2.

2. **Case $\beta \neq 1$:** We use the standard Euler approximation:

$$S(t+\delta t) = S(t) + \mu S(t)\delta t + \alpha(t) S(t)^{\beta}\sqrt{\delta t}\, Z_1. \tag{15}$$

   To ensure positivity, we apply the constraint:

$$S(t+\delta t) \leftarrow \max\left(S(t+\delta t), 0\right). \tag{16}$$

3.2 Generating Correlated Brownian Motions

To generate the correlated random variables Z_1 and Z_2 with correlation ρ, we use the *Cholesky decomposition* method [Glasserman, 2004]. Starting with two independent standard normal random variables $Z_1, Z_2 \sim \mathcal{N}(0,1)$, we construct:

$$\begin{pmatrix} Z_1 \\ Z_2^c \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix}, \tag{17}$$

which gives:

$$Z_2^c = \rho Z_1 + \sqrt{1-\rho^2}\, Z_2. \tag{18}$$

By construction, Z_1 and Z_2^c are standard normal with $\mathrm{Corr}(Z_1, Z_2^c) = \rho$. We then use Z_1 in equations (14) and (15), and Z_2^c in equation (13).

3.3 Algorithm Summary

The complete simulation algorithm is as follows:

1. Fix $\delta t = T/n$.
2. Start with $S(0) \leftarrow S_0$, $\alpha(0) \leftarrow \alpha_0$, and $t \leftarrow 0$.
3. Repeat for $i = 1, \ldots, n$:
   (a) Generate $Z_1, Z_2 \sim \mathcal{N}(0,1)$ independently.

(b) Compute $Z_2^c = \rho Z_1 + \sqrt{1-\rho^2}\, Z_2$.

(c) Update $\alpha(t+\delta t)$ using equation (13).

(d) Update $S(t+\delta t)$ using equation (14) (if $\beta = 1$) or equation (15) (if $\beta \neq 1$).

(e) If using equation (15), apply constraint (16).

(f) Set $t \leftarrow t + \delta t$.

4 Implementation in R

We now implement the SABR model in R following the numerical scheme described in Section 3. Our implementation consists of four main functions that handle different aspects of the simulation.

4.1 Function 1: Geometric Brownian Motion

For comparison purposes, we first implement a standard geometric Brownian motion path generator. This will allow us to verify that the SABR model reduces to the Black-Scholes model when $\beta = 1$ and $\nu = 0$.

```
> # This does not fill into our SABR model since SABR is not GBM.
> # But this function will be used to compare SABR and BS.
> # SABR reduces to BS when v = 0 and beta = 1 which we will show later.
>
> # Inputs:
> #  --- S0: initial price
> #  --- mu: drift parameter
> #  --- sigma: volatility
> #  --- n_steps: time steps (as n approaches inf we move to continuous time)
> #  --- H: Exercise time horizon
>
> # How we get to discretized formula:
>
> # By methods of stochastic calculus (Ito's lemma), we get the exact solution
> # to the Black-Scholes SDE: dS = mu*S*dt + sigma*S*dWt
> # Apply Ito's Lemma to ln(S):
> # --- d(ln S) = (mu - 0.5*sigma^2)dt + sigma*dWt
> # Discretize the change in log-price over step dt:
> # --- ln(S(t+1)/S(t)) = (mu - 0.5*sigma^2)*dt + sigma*sqrt(dt)*Z
> # Take the exponent of both sides and multiply both sides by S(t):
> # --- S(t+1) = S(t) * exp( (mu - 0.5*sigma^2)*dt + sigma*sqrt(dt)*Z )
> # This ensures that S(t) > 0.
>
> brownian.path <- function(S0, drift, sigma, n_steps, H, Z = NULL){
+
+   # ensure all inputs given
+   if (missing(S0) || missing(drift) || missing(sigma) ||
+       missing(n_steps) || missing(H)) {
+         stop("missing one or more required inputs")}
+
+   # ensure correct types
+   if (!is.numeric(S0) || !is.numeric(drift) ||
+       !is.numeric(sigma) || !is.numeric(H)) {
+         stop("S0, drift, sigma, and H must be numeric")}
+
+   # ensure input logic
+   if (S0 <= 0) stop("Initial price S0 must be positive")
+   if (H <= 0) stop("Time horizon H must be positive")
+   if (sigma < 0) stop("Volatility sigma cannot be negative")
```

```
+
+   # ensure n_steps integer
+   if (n_steps <= 0 || n_steps %% 1 != 0) {
+     stop("Error: n_steps must be a positive integer.")}
+
+   # Set size of time steps (approaches 0 as N-steps approaches inf)
+   dt <- H / n_steps
+
+   # To do a proper comparison later on with the same randomness
+   # we need an argument such that brownian.path can accept Z1.
+   # Generate n_steps standard normal random variables
+   if (is.null(Z)) Z <- rnorm(n_steps)
+   if (length(Z) != n_steps) stop("Z must have length n_steps")
+
+
+   # We use GBM - ensuring S > 0 and giving true solution under const. vol.
+   # S_t = S_0 * exp( (mu - 0.5*sigma^2)*t + sigma*W_t )
+   log_increments <- (drift - 0.5 * sigma^2) * dt + sigma * sqrt(dt) * Z
+
+   # Cumsum to get the path of the exponent
+   # we add 0 so the first element is S0 * exp(0) = S0
+   log_path <- cumsum(c(0, log_increments))
+
+   # Exponentiation of both sides of the log_path above
+   S <- S0 * exp(log_path)
+
+   return(S)
+ }
```

This function implements equation (6). The optional argument `Z` allows us to pass pre-generated random numbers, which is useful for comparing paths with the same randomness.

4.2 Function 2: Generating Correlated Shocks

This function generates two correlated standard normal random vectors using the Cholesky decomposition method described in equation (17).

```
> # In the SABR model, one of the key parameters is rho. Which represents the
> # the correlation between the movement of the price path and the volatility path.
>
> # We set this parameter as an argument, and then generate a second set of
> # random shocks (using Choleksy) that's correlated to the first with level rho.
>
> # In reality, this would represent the asymmetry of the volatility smile.
>
> # Inputs:
> #  --- rho: we set the size of the correlation of the two random variables
> #  --- n_steps: time steps (as n approaches inf we move to continuous time)
> #  --- H: Exercise time horizon
>
> sabr.CorrShocks <- function(rho, n_steps, H) {
+
+   if (missing(rho) || missing(n_steps) || missing(H)) {
+     stop("missing one or more required inputs")}
+
+   if (!is.numeric(rho) || !is.numeric(n_steps) || !is.numeric(H)) {
+     stop("must have numeric inputs")}
```

```
+
+   if (n_steps <= 0 || n_steps %% 1 != 0)
+     stop("n_steps must be a positive integer")
+
+   if (rho < -1 || rho > 1)  stop("rho must be between -1 and 1")
+   if (H <= 0) stop(" time horizon H must be positive")
+
+   dt <- H / n_steps
+
+   Z1  <- rnorm(n_steps)
+   Z2  <- rnorm(n_steps)
+
+   # We implement the correlation of the two shocks using the Cholesky Matrix
+   # Below is just the linear solution to dW2.
+   Z2c <- rho * Z1 + sqrt(1 - rho^2) * Z2
+   out <- list(dt  = dt, Z1  = Z1, Z2  = Z2, Z2c = Z2c)
+
+   return(out)
+ }
```

The function returns a list containing the time step `dt`, the independent shocks `Z1` and `Z2`, and the correlated shock `Z2c`. The correlation can be verified using R's built-in `cor()` function.

4.3 Function 3: SABR Volatility Path

This function simulates the stochastic volatility process $\alpha(t)$ using equation (13).

```
> # Purpose: Simulate SABR volatility level alpha(t):
> # --- d alpha_t = v * alpha_t * dW2_t   (lognormal, no drift)
> # Positivity: Uses exact/lognormal update:
> # --- alpha_{t+dt} = alpha_t * exp(-0.5*nu^2*dt + nu*sqrt(dt)*Z2c_t)
>
> # In the code we handle two cases explicitly:
> # (1) Volaitlity is constant i.e) v ==0
> # (2) Volatility is stochastic
>
> # For case (1): It would be trivial to apply the formula for each step, hence
> # for computational efficiency, if v == 0 we simply generate a vector where the
> # volatility is set to alpha(0) for N + 1 times.
>
> # For case (2): we implement the discretized version of the volatility formula
> # Using SABR models volatility SDE take log of both sides and then Ito:
> # --- log[a(t+dt)] = log[a(t)] - (1/2)(v^2)dt + v*sqrt(dt)*Z2c
> # Exponentiating both sides gives:
> # ---- a(t+dt) = a(t)*exp(- (1/2)(v^2)dt + v*sqrt(dt)*Z2c)
> # Then we have:
> # ---- drift term: - (1/2)(v^2)dt
> # ---- diff scale: v*sqrt(dt)*Z2c
>
> # Inputs:
> #  --- alpha0: initial alpha(0)
> #  --- v: vol-of-vol
> #  --- n: number of time steps
> #  --- H: horizon in years
> #
> # Best usage would be to pass Z2c from sabr.shocks()
> #
```

```
> # Optionally if Z2c is not provided we can compute it interrnally
> #   -- rho : correlation in [-1, 1]
> #   -- Z1  : vector length n (standard normals)
> #   -- Z2  : vector length n (standard normals, iid)
> #
> # Results: numeric vector length n+1, alpha path including alpha0
>
> sabr.AlphaPath <- function(alpha0, v, n_steps, H, Z2c) {
+
+   if (missing(alpha0) || missing(v) ||
+       missing(n_steps) || missing(H)) {
+     stop("missing one or more required inputs")}
+
+   if (!is.numeric(v) || !is.numeric(alpha0) ||
+       !is.numeric(n_steps) || !is.numeric(H)) {
+     stop("must have numeric inputs")}
+
+   if (alpha0 <= 0) stop("initial volatility alpha0 must be positive")
+   if (v < 0) stop("volatility of volatility nu cannot be negative")
+   if (H <= 0) stop(" time horizon H must be positive")
+   if (n_steps <= 0 || n_steps %% 1 != 0) stop("n_steps must be positive integer")
+
+
+   if (missing(Z2c) || is.null(Z2c)) {
+     stop("must provide a vector of correlated random shocks Z2c.")}
+
+   if (length(Z2c) != n_steps) stop("length of Z must match n_steps")
+
+   dt <- H / n_steps
+
+   # CASE 1: constant volatility (separated to improve computational efficiency)
+   if (v == 0) { return(rep(alpha0, n_steps + 1))  }
+
+   # CASE 2: stochastic volatility
+   drift_term      <- -0.5 * v^2 * dt
+   diff_scale      <-  v * sqrt(dt) * Z2c
+   log_increments  <- drift_term + diff_scale
+   log_path        <- cumsum(c(0, log_increments))
+   alpha_path      <- alpha0 * exp(log_path)
+
+   return(alpha_path)
+ }
>
```

The function handles the special case $\nu = 0$ efficiently by simply returning a constant volatility path. This ensures that $\alpha(t) > 0$ for all time steps.

4.4 Function 4: SABR Price Path

This is the main function that simulates the SABR forward price process using equations (14) and (15).

```
> # Cannot simulate the SABR price path exactly hence we approximate it discretely
> # continuous time SABR SDE given as:
> #   -- dS(t) = drift*S(t)*dt + a(t)*S(t)^B*dWt
>
> # We handle two cases for beta, specifically beta = 1 and beta != 1:
>
```

```
> # (1) For beta == 1: The discretized formula then reduces to BLack Scholes
> # we start with dS(t) = S(t)* (drift*dt + a(t)*dWt)
> # taking log of both sides:
> # -- log(S(t+dt)) = log(S(t)) + (drift-(1/2)*a(t))*dt + a(t)*sqrt(dt)*Z(t)
> # taking exponent of both side:
> # -- S(t+dt) = S(t) * exp(drift-0.5*alpha[i]^2)*dt + a(t)*sqrt(dt)*Z1[i])
>
>
> # (2) For beta != 1:
> #   -- If beta != 1 the formual has its own discretized Hagan approximation
>
> #Inputs:
> # -- S0: initial price
> # -- alpha(0): intial volatility
> # -- drift: drift parameter
> # -- beta: price elasticity
> # -- v: volatility of the volatility
> # -- rho: the correlation between the brownian shocks.
> # -- n_steps: number of simulations
> # -- H : time horizon
> # -- Z1, Z2c: correlated brownian shocks with correlation rho
>
> sabr.path <- function(S0, alpha0, drift, beta, v, rho, n_steps, H,
+                       Z1 = NULL, Z2c = NULL, return_alpha = TRUE) {
+
+   if (missing(S0) || missing(alpha0) || missing(drift) || missing(beta) ||
+       missing(v) || missing(rho) || missing(n_steps) || missing(H)) {
+     stop("missing one or more required inputs")
+   }
+
+   if (!is.numeric(S0) || !is.numeric(alpha0) || !is.numeric(drift) ||
+       !is.numeric(beta) || !is.numeric(v) || !is.numeric(rho) ||
+       !is.numeric(n_steps) || !is.numeric(H)) {
+     stop("must have numeric inputs")
+   }
+
+   if (beta < 0 || beta > 1) stop("beta must be between 0 and 1")
+
+   dt <- H / n_steps
+
+   # random shocks generated if they are not supplied
+   if (is.null(Z1) && is.null(Z2c)) {
+     shocks <- sabr.CorrShocks(rho = rho, n_steps = n_steps, H = H)
+     Z1  <- shocks$Z1
+     Z2c <- shocks$Z2c
+
+   } else if (is.null(Z1) || is.null(Z2c)) {
+     stop("Provide both Z1 and Z2c, or neither.")
+
+   } else { if (length(Z1) != n_steps || length(Z2c) != n_steps)
+     stop("Z1 and Z2c must each have length n_steps.")
+   }
+
+   # Volatility path using the helper function created earlier
+   alpha <- sabr.AlphaPath(alpha0, v, n_steps, H, Z2c)
+
```

```
+   # Now setting up the price path.
+   S <- numeric(n_steps + 1)
+   S[1] <- S0
+
+   # Case (1): For beta == 1
+   if (beta == 1) {
+
+       log_increments <- (drift - 0.5 * alpha[1:n_steps]^2) * dt +
+                          alpha[1:n_steps] * sqrt(dt) * Z1
+       S <- S0 * exp(c(0, cumsum(log_increments)))
+
+   # Case (2): For beta != 1
+   } else {
+     for (i in 1:n_steps) {
+           S_current <- S[i]
+           diffusion <- alpha[i] * (max(S_current, 0))^beta * sqrt(dt) * Z1[i]
+           S_next <- S_current + drift * S_current * dt + diffusion
+           S[i + 1] <- max(S_next, 0) # positivity preserving
+     }
+   }
+   if (return_alpha) { list(S = S, alpha = alpha) } else { S }
+ }
```

This function implements the complete SABR simulation algorithm. It automatically generates correlated shocks if they are not provided. The function returns both the price path $S(t)$ and the volatility path $\alpha(t)$ when `return_alpha = TRUE`.

4.5 Function 5: Pricing European call with Monte Carlo simulation

Computes the value of a European call option with the SABR model using Monte Carlo simulation with antithetic random numbers.

```
> # Purpose: Computes the value of a European call option using the SABR model
> #          using Monte Carlo simulation with antithetic random numbers for variance reduction.
> #          The function simulates risk-neutral price paths to estimate the discounted expected p
> #          the European Call
> #          V(0) = exp(-rT) * E[max(S(T) - E, 0)].
> #
> # Inputs:
> #   -- S0      : Initial price (S0 > 0)
> #   -- alpha0  : Initial alpha (alpha0 > 0)
> #   -- E       : strike price (E >= 0)
> #   -- r       : Risk-free interest rate (decimal > 0)
> #   -- beta    : Elasticity parameter (0 <= beta <= 1)
> #   -- nu      : vol-of-vol >= 0
> #   -- rho     : Correlation in [-1;1]
> #   -- H       : Time to expiration in years (H > 0)
> #   -- n_steps : number of time steps (integer >= 1)
> #   -- N       : Total number of simulation paths (even integer because of antithetics)
> #
>
> #
> # Output:
> #   A list containing:
> #   -- $value    : price of European call
> #   -- $confint  :  numeric  vector with lower and upper bound of 95% conf. interval
> #   -- $stderror : standard error of the price
```

```
>
>
> sabr.europ.mc <- function(S0, alpha0, E, r, beta, v, rho, H, n_steps, N) {
+
+   if (missing(S0) || missing(alpha0) || missing(E) || missing(r) ||
+       missing(beta) || missing(v) || missing(rho) || missing(H) ||
+       missing(n_steps) || missing(N)) {
+     stop("missing one or more required inputs")
+   }
+
+   if (!all(
+     sapply(list(S0, alpha0, E, r, beta, v, rho, H, n_steps, N), is.numeric))) {
+       stop("all inputs must be numeric.")
+   }
+
+   if (S0 <= 0 || alpha0 <= 0 || E < 0)
+     stop("S0, alpha0, and E must be positive.")
+   if (H <= 0 || n_steps <= 0 || N <= 0)
+     stop("H, n_steps, and N must be positive.")
+   if (beta < 0 || beta > 1) stop("beta must be between 0 and 1.")
+   if (rho < -1 || rho > 1) stop("rho must be between -1 and 1.")
+
+   if (N %% 2 != 0) stop("N must be even for antithetic sampling")
+
+   #pair_payoffs creates N/2 pairs Z and -Z and returns the
+   #average payoff of each pair
+   pair_payoffs <- replicate(N / 2, {
+
+     # generate shocks (by calling sabr.CorrShocks )
+     shocks <- sabr.CorrShocks(rho, n_steps, H)
+     Z1  <- shocks$Z1
+     Z2c <- shocks$Z2c
+
+     # use random numbers Z
+     path_positive <- sabr.path(S0, alpha0, r, beta, v, rho, n_steps, H,
+                                Z1 = Z1, Z2c = Z2c, return_alpha = FALSE)
+     S_T_positive <- tail(path_positive, 1) # we only need the final price
+
+     # use antithetic random number. We must negate BOTH Z1 and Z2c
+     # correlation is preserved because corr(-X, -Y) = corr(X, Y)
+     path_negative <- sabr.path(S0, alpha0, r, beta, v, rho, n_steps, H,
+                                Z1 = -Z1, Z2c = -Z2c, return_alpha = FALSE)
+     S_T_negative <- tail(path_negative, 1)
+
+
+     payoff_positive <- max(S_T_positive - E, 0)
+     payoff_negative <- max(S_T_negative - E, 0)
+
+     # return the average of the pair
+     return((payoff_positive + payoff_negative) / 2)
+   })
+
+   # discount the average payoffs from the pairs
+   discounted_payoffs <- exp(-r * H) * pair_payoffs
+
+   # generated statistics as demanded in write-up
```

```
+   # standard error is calculated based on the N/2 pairs,
+   #which reduces variance.
+
+   price <- mean(discounted_payoffs)
+   standard_error <- sd(discounted_payoffs) / sqrt(N / 2)
+
+   return(list(
+     value = price,
+     confint = c(price - 1.96 * standard_error, price + 1.96 * standard_error),
+     std_error = standard_error
+   ))
+ }
>
```

The function generates N/2 random numbers and N/2 antithetics thereof. Then it creates two price paths for each pair.In every simulation, the function returns the average of the call payoffs of each pair of prices at maturity $S(T)$.This method reduces the variance of the estimation.

The function returns the option price, the bounds of the 0.95 confidence interval and the standard error of the price estimate.

4.6 Function 6: Pricing European call with Black-Scholes

This function computes price of European call using the standard Black-Scholes formula.

```
> # Purpose: computes price of European call using standard Black-Scholes formula
> #          C = S * N(d1) - E * exp(-rT) * N(d2)
> #
> # Inputs:
> #   -- S     : initial stock price (S > 0)
> #   -- E     : strike pirce (E >= 0)
> #   -- r     : risk-free interest rate (decimal, e.g., 0.05)
> #   -- sigma : volatility of underlying (sigma >= 0)
> #   -- T     : time to expiration in years (T > 0)
> #
> # Output:
> #   -- value : price of European call.
>
> bs.europ.formula <- function(S0, E, r, sigma, H) {
+
+   if (missing(S0) || missing(E) || missing(r) || missing(sigma) || missing(H)) {
+     stop("missing one or more required inputs")
+   }
+
+   if (!all(sapply(list(S0, E, r, sigma, H), is.numeric))) {
+     stop("all inputs must be numeric.")
+   }
+
+   if (S0 <= 0) stop("stock price S must be positive.")
+   if (E < 0)  stop("strike price E cannot be negative.")
+   if (sigma < 0) stop("volatility sigma cannot be negative.")
+   if (H <= 0) stop("time to expiration H must be positive.")
+
+   d1 <- (log(S0 / E) + (r + 0.5 * sigma^2) * H) / (sigma * sqrt(H))
+   d2 <- d1 - sigma * sqrt(H)
+   price <- S0 * pnorm(d1) - E * exp(-r * H) * pnorm(d2)
+
+   return(price)
```

```
+ }
>
```

If the Monte Carlo implementation of the SABR model is correct, the option price from the Black-Scholes formula should equal the price from `sabr.europ.mc` under the Black-Scholes conditions ($\beta = 1$, $\nu = 0$, $\alpha_0 = \sigma$).

5 Numerical Experiments

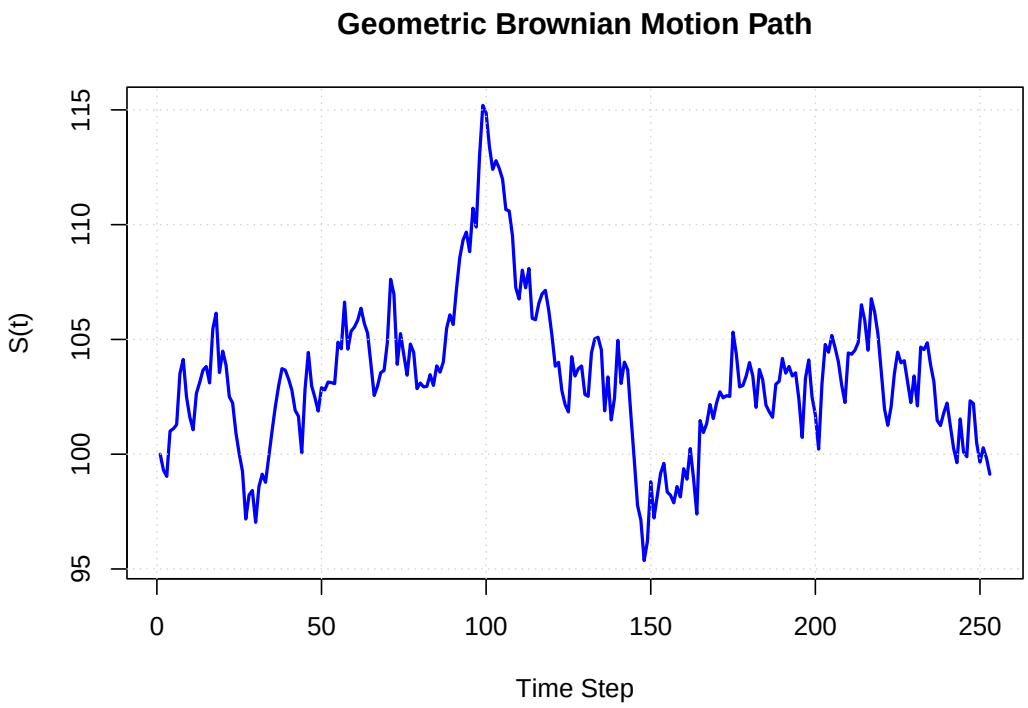
We now conduct several numerical experiments to verify our implementation and explore the behavior of the SABR model under different parameter configurations.

```
> set.seed(123)  # For reproducibility
```

5.1 Experiment 1: Geometric Brownian Motion Path

We first generate a sample path for a standard geometric Brownian motion using the `brownian.path` function.

```
> Test_BrownianPath <- brownian.path(100, 0.05, 0.2, 252, 1)
> plot(Test_BrownianPath, type = 'l', lwd = 2, col = "blue",
+      main = "Geometric Brownian Motion Path",
+      xlab = "Time Step", ylab = "S(t)")
> grid()
```



The figure shows a typical realization of a geometric Brownian motion with initial price $S_0 = 100$, drift $\mu = 0.05$, volatility $\sigma = 0.2$, over one year with 252 time steps.

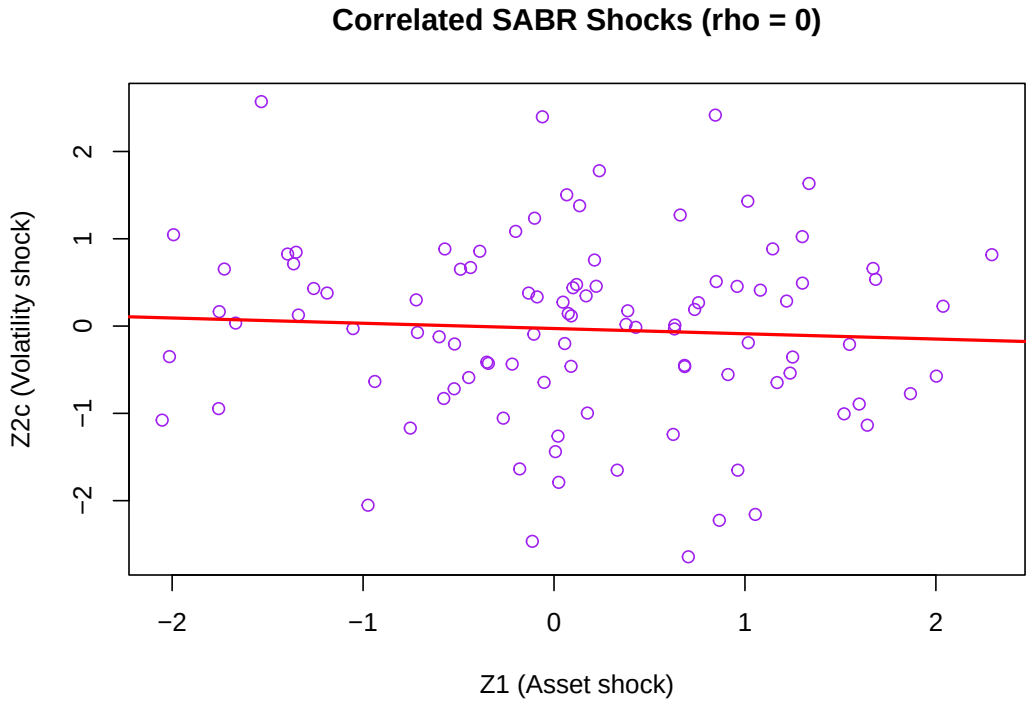
5.2 Experiment 2: Verifying Correlation

We verify that the `sabr.CorrShocks` function correctly generates correlated random variables. We test the correlation between Z_1 and Z_2^c using three different methods for rho $\rho \in \{0, 0.1, 0.5, 1.0\}$ The three methods used are as follows:

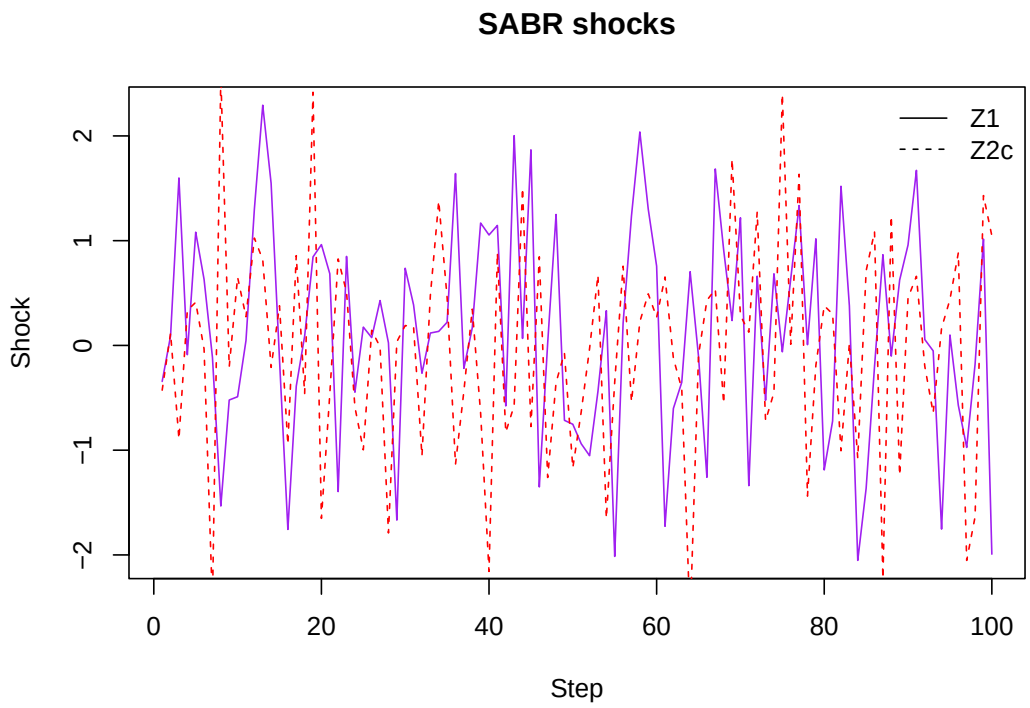
- Using the BaseR::cor. The results should be in the vicinity of the rho set.
- Using a scatter plot with a regression line. The slope of the regression line should be in vicinity of rho

- Visualize using time series data, although correlation is not obvious point wise.

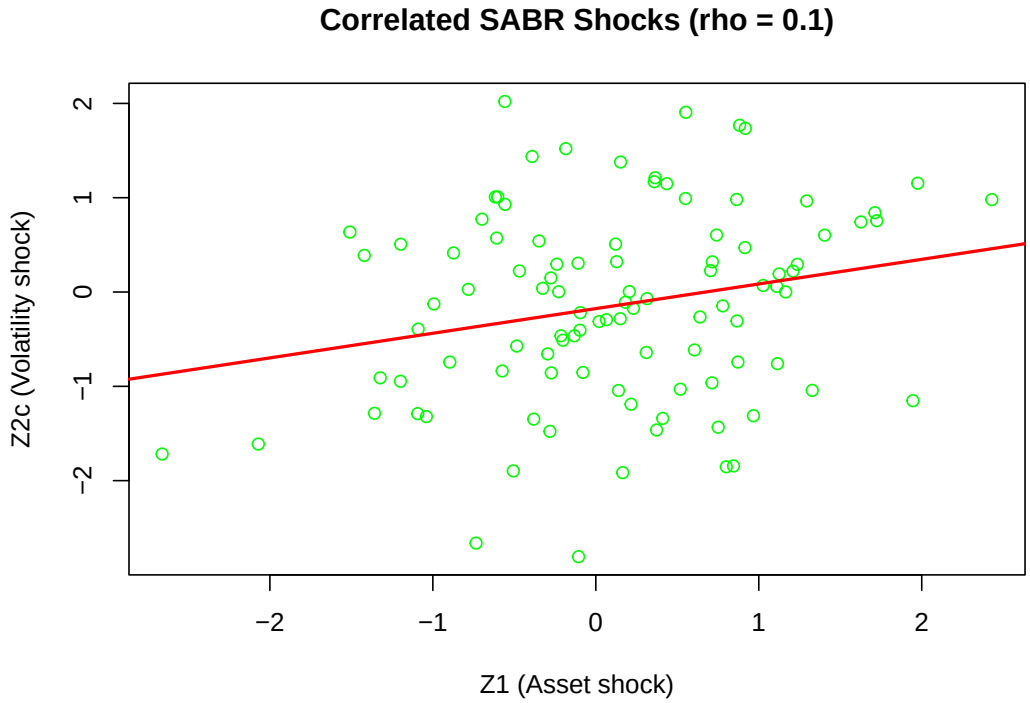
```
> Test_sabr.CorrShocks_00 <- sabr.CorrShocks(0, 100, 1)
> # (1)
> cor(Test_sabr.CorrShocks_00$Z1, Test_sabr.CorrShocks_00$Z2c)

[1] -0.05945955

> #(2)
> Regr_00 <- lm(Z2c ~ Z1, data = as.data.frame(Test_sabr.CorrShocks_00))
> plot(Test_sabr.CorrShocks_00$Z1, Test_sabr.CorrShocks_00$Z2c,
+      xlab = "Z1 (Asset shock)", ylab = "Z2c (Volatility shock)",
+      main = "Correlated SABR Shocks (rho = 0)", col  = "purple")
> abline(Regr_00, col = "red", lwd = 2)
```



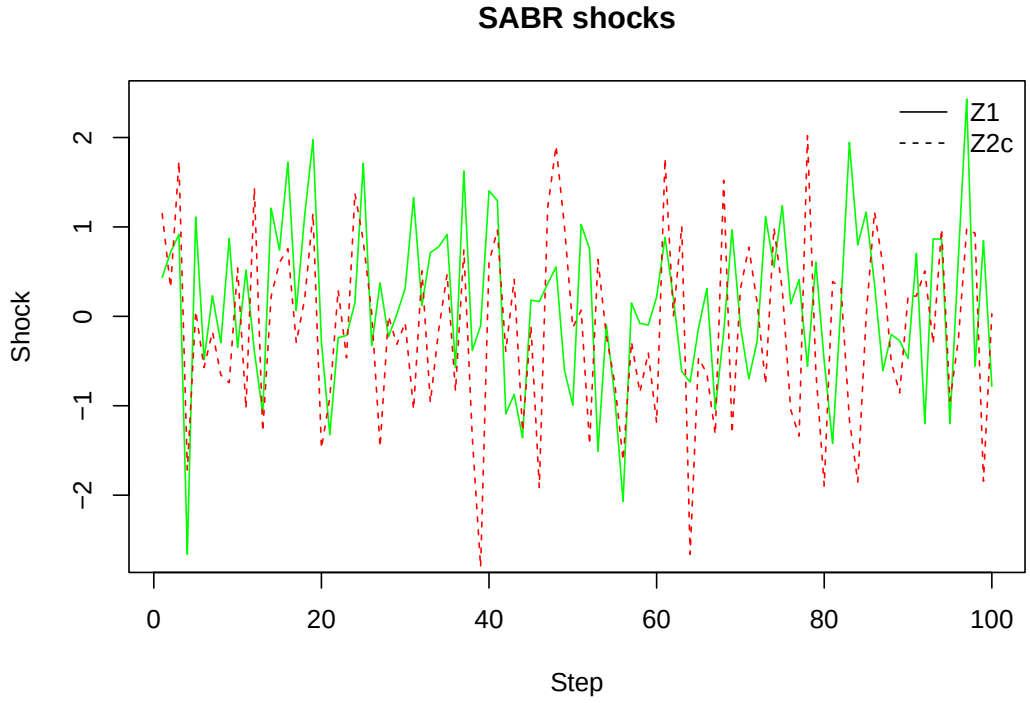

```
> # (3)
> plot(Test_sabr.CorrShocks_00$Z1, type = "l", col = "purple",
+      xlab = "Step", ylab = "Shock", main = "SABR shocks")
> lines(Test_sabr.CorrShocks_00$Z2c, lty = 2, col = "red")
> legend("topright", legend = c("Z1", "Z2c"), lty = c(1, 2), bty = "n")
```



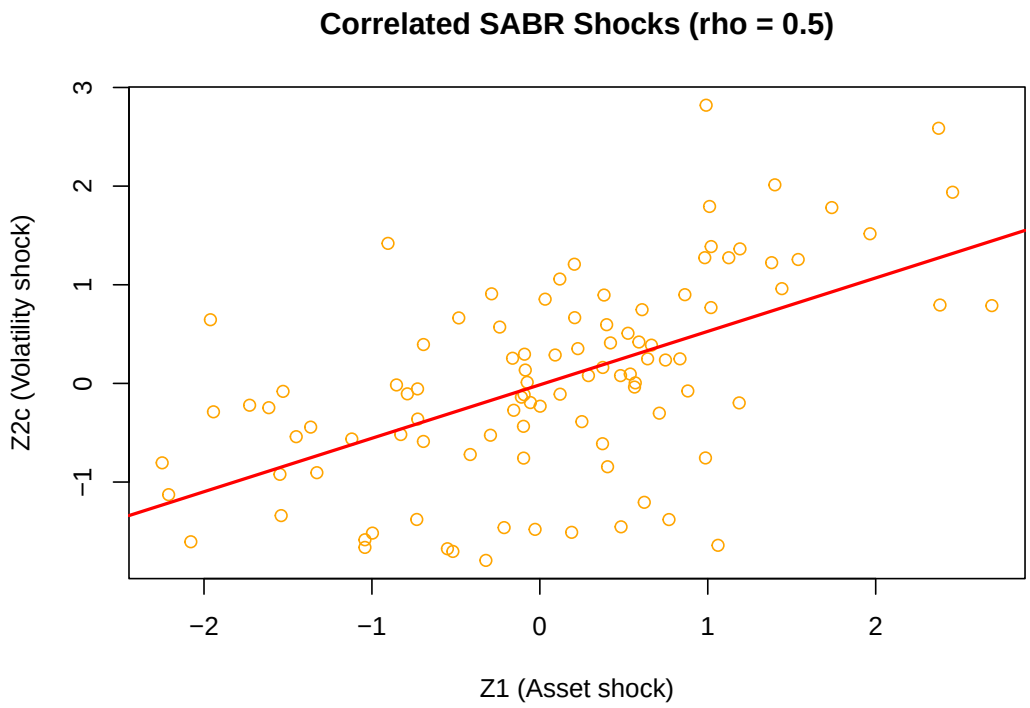

```
> Test_sabr.CorrShocks_01 <- sabr.CorrShocks(0.1, 100, 1)
> # (1)
> cor(Test_sabr.CorrShocks_01$Z1, Test_sabr.CorrShocks_01$Z2c)

[1] 0.232764

> # (2)
> Regr_01 <- lm(Z2c ~ Z1, data = as.data.frame(Test_sabr.CorrShocks_01))
> plot(Test_sabr.CorrShocks_01$Z1, Test_sabr.CorrShocks_01$Z2c,
+      xlab = "Z1 (Asset shock)", ylab = "Z2c (Volatility shock)",
+      main = "Correlated SABR Shocks (rho = 0.1)", col  = "green")
> abline(Regr_01, col = "red", lwd = 2)
```



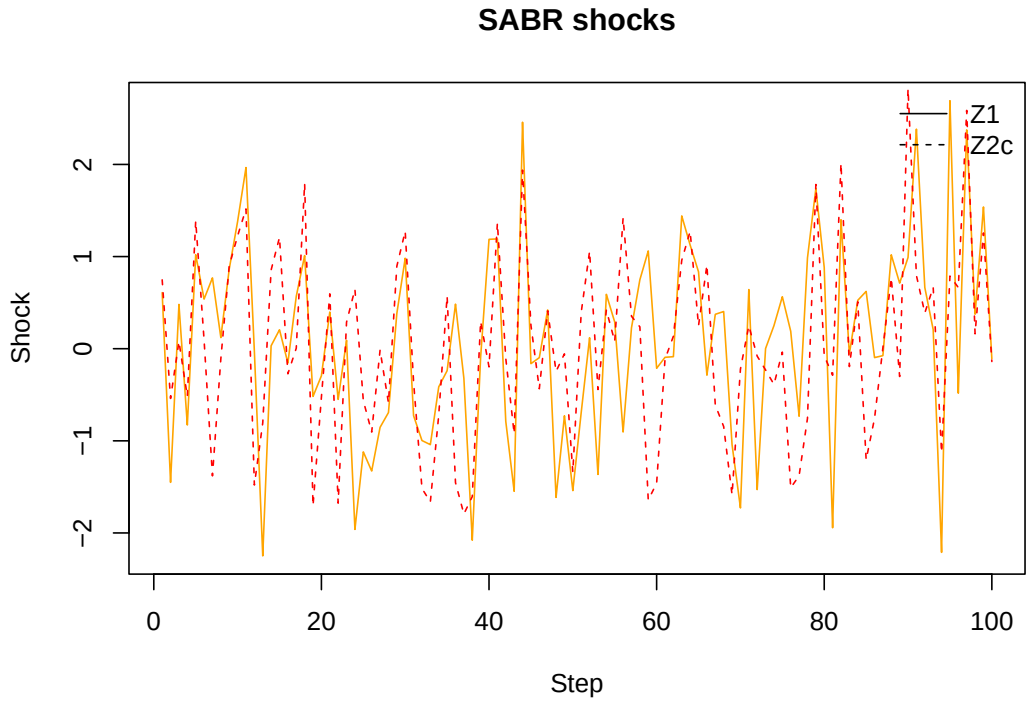
```
> # (3)
> plot(Test_sabr.CorrShocks_01$Z1, type = "l", col = "green",
+      xlab = "Step", ylab = "Shock", main = "SABR shocks")
> lines(Test_sabr.CorrShocks_01$Z2c, lty = 2, col = "red")
> legend("topright", legend = c("Z1", "Z2c"), lty = c(1, 2), bty = "n")
```




```
> Test_sabr.CorrShocks_05 <- sabr.CorrShocks(0.5, 100, 1)
> # (1)
> cor(Test_sabr.CorrShocks_05$Z1, Test_sabr.CorrShocks_05$Z2c)

[1] 0.5639312

> # (2)
> Regr_05 <- lm(Z2c ~ Z1, data = as.data.frame(Test_sabr.CorrShocks_05))
> plot(Test_sabr.CorrShocks_05$Z1, Test_sabr.CorrShocks_05$Z2c,
+      xlab = "Z1 (Asset shock)", ylab = "Z2c (Volatility shock)",
+      main = "Correlated SABR Shocks (rho = 0.5)", col  = "orange")
> abline(Regr_05, col = "red", lwd = 2)
```

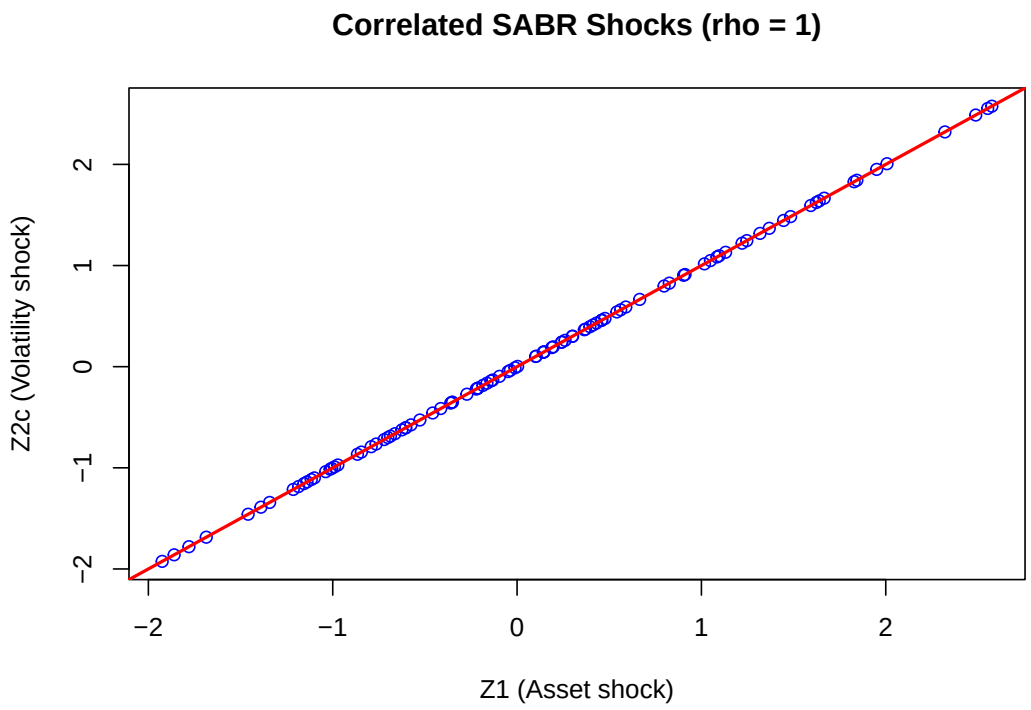



```
> # (3)
> plot(Test_sabr.CorrShocks_05$Z1, type = "l", col = "orange",
+      xlab = "Step", ylab = "Shock", main = "SABR shocks")
> lines(Test_sabr.CorrShocks_05$Z2c, lty = 2, col = "red")
> legend("topright", legend = c("Z1", "Z2c"), lty = c(1, 2), bty = "n")
```

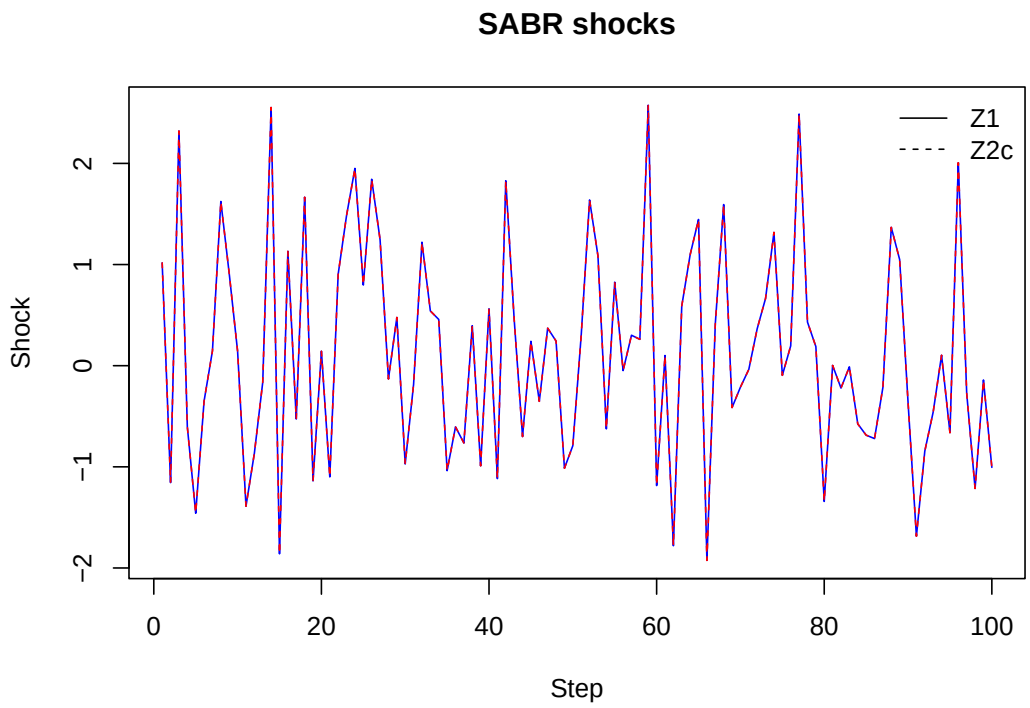



```
> Test_sabr.CorrShocks_10 <- sabr.CorrShocks(1, 100, 1)
> # (1)
> cor(Test_sabr.CorrShocks_10$Z1, Test_sabr.CorrShocks_10$Z2c)

[1] 1
```

```
> # (2)
> Regr_10 <- lm(Z2c ~ Z1, data = as.data.frame(Test_sabr.CorrShocks_10))
> plot(Test_sabr.CorrShocks_10$Z1, Test_sabr.CorrShocks_10$Z2c,
+      xlab = "Z1 (Asset shock)", ylab = "Z2c (Volatility shock)",
+      main = "Correlated SABR Shocks (rho = 1)", col  = "blue")
> abline(Regr_10, col = "red", lwd = 2)
```



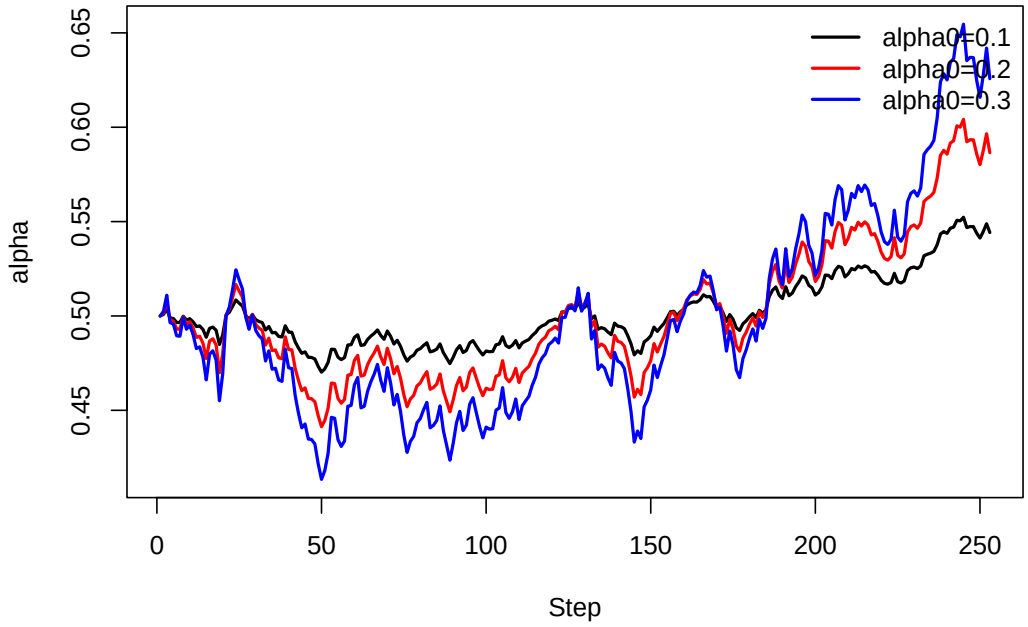

```
> # (3)
> plot(Test_sabr.CorrShocks_10$Z1, type = "l", col = "blue",
+      xlab = "Step", ylab = "Shock", main = "SABR shocks")
> lines(Test_sabr.CorrShocks_10$Z2c, lty = 2, col = "red")
> legend("topright", legend = c("Z1", "Z2c"), lty = c(1, 2), bty = "n")
>
```



5.3 Experiment 3: SABR Volatility Path

We generate and visualize three sample paths of the stochastic volatility process $\alpha(t)$. All three paths are set to start at $\alpha(0) = 0.5$ with varying volatility-of-volatility (v) values. The path remains strictly positive due to the lognormal formulation in equation (13).

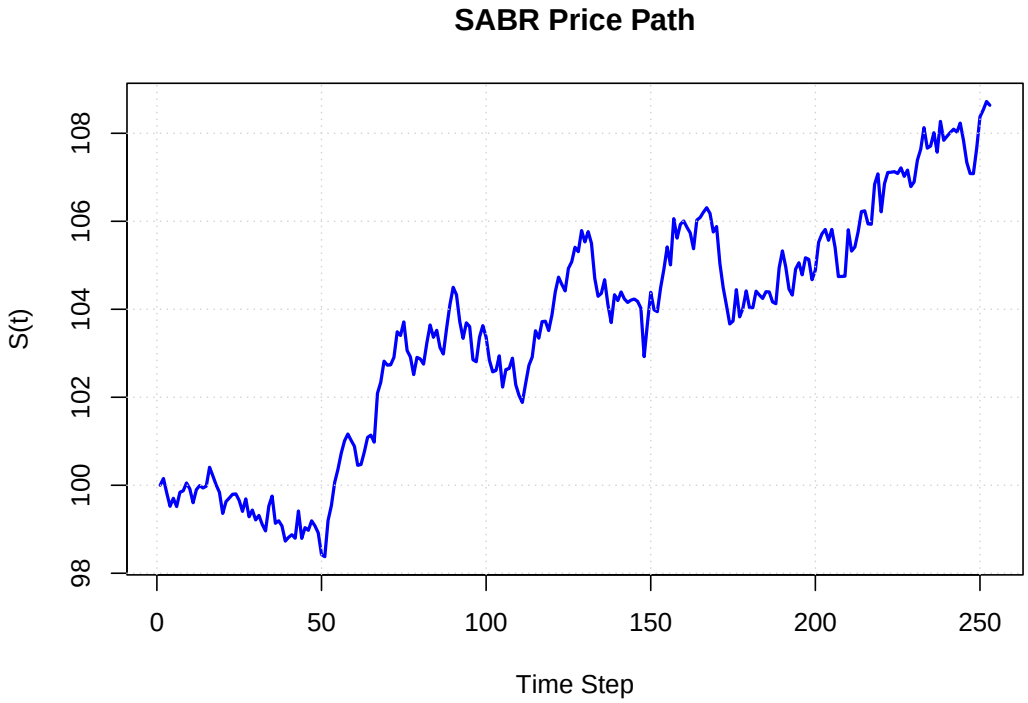
```
> shocks <- sabr.CorrShocks( rho = -0.3, n_steps = 252, H = 1)
> Path1 <- sabr.AlphaPath(alpha0 = 0.5, v = 0.1, n = 252, H = 1, Z2c = shocks$Z2c)
> Path2 <- sabr.AlphaPath(alpha0 = 0.5, v = 0.2, n = 252, H = 1, Z2c = shocks$Z2c)
> Path3 <- sabr.AlphaPath(alpha0 = 0.5, v = 0.3, n = 252, H = 1, Z2c = shocks$Z2c)
> ylim <- range(c(Path1, Path2, Path3), finite = TRUE)
> plot(Path1, type = "l", ylim = ylim, col = "black",
+      lwd = 2, xlab = "Step", ylab = "alpha")
> lines(Path2, col = "red",  lwd = 2)
> lines(Path3, col = "blue", lwd = 2)
> legend("topright", legend = c("alpha0=0.1", "alpha0=0.2", "alpha0=0.3"),
+        col = c("black", "red", "blue"), lty = 1, lwd = 2, bty = "n")
>
```



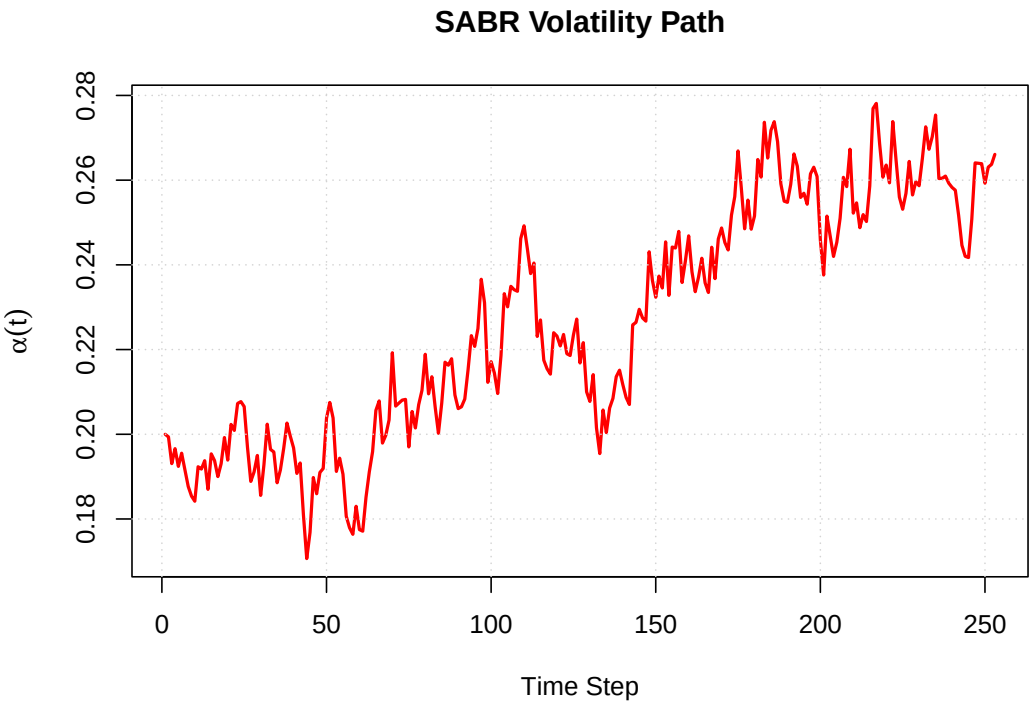
5.4 Experiment 4: SABR Price and Volatility Paths

We now simulate the complete SABR model, generating both price and volatility paths simultaneously.

```
> shocks <- sabr.CorrShocks(rho = -0.3, n_steps = 252, H = 1)
> out <- sabr.path(S0 = 100, alpha0 = 0.2, drift = 0.05, beta = 0.7,
+                  v = 0.5, rho = -0.3, n_steps = 252, H = 1,
+                  Z1 = shocks$Z1, Z2c = shocks$Z2c, return_alpha = TRUE)
> plot(out$S, type = "l", lwd = 2, col = "blue",
+      main = "SABR Price Path",
+      xlab = "Time Step", ylab = "S(t)")
> grid()
```



```
> plot(out$alpha, type = "l", lwd = 2, col = "red",
+      main = "SABR Volatility Path",
+      xlab = "Time Step", ylab = expression(alpha(t)))
> grid()
```



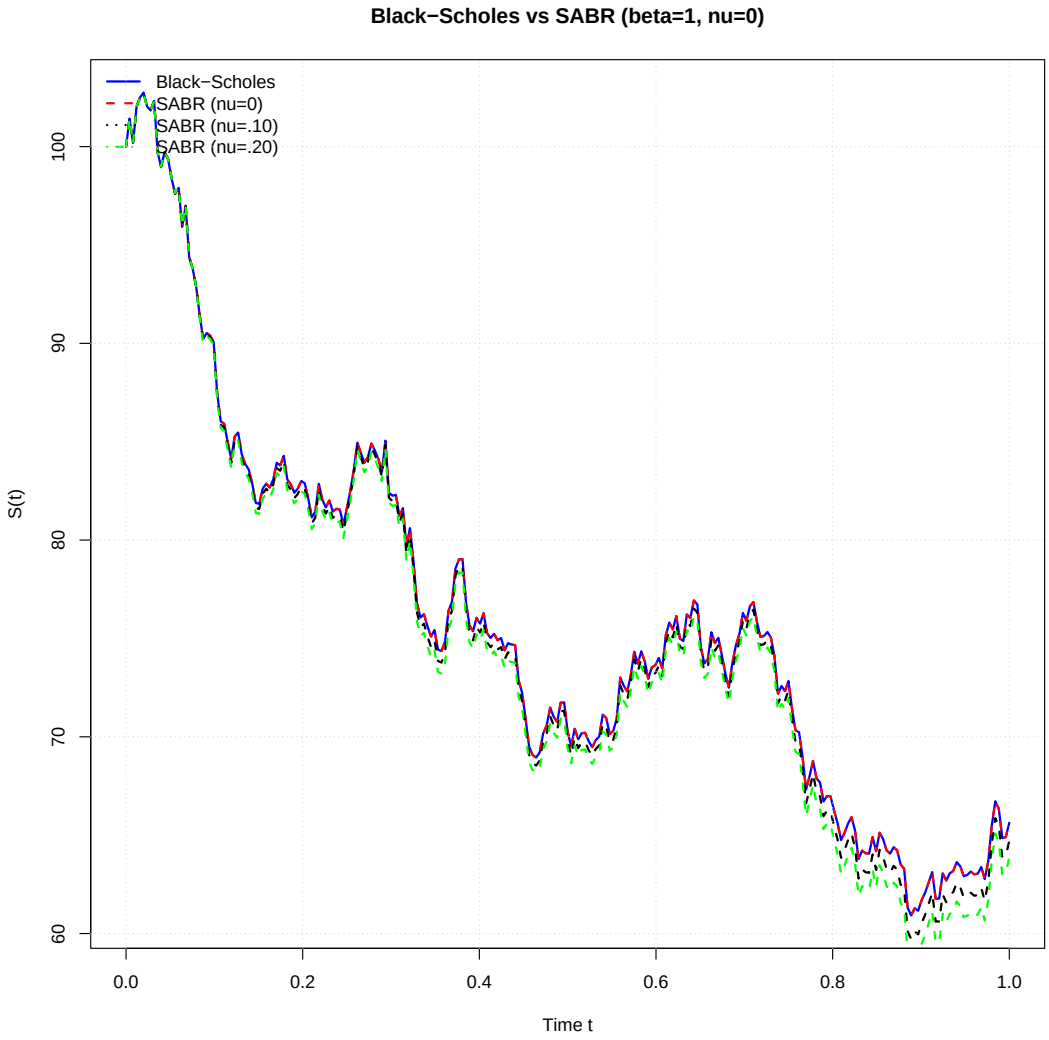
The first figure shows a SABR price path with $\beta = 0.7$ (CEV-type behavior), while the second figure displays the corresponding stochastic volatility path. The negative correlation $\rho = -0.3$ captures the leverage effect: periods of declining prices tend to coincide with increasing volatility.

5.5 Experiment 5: Comparison with Black-Scholes

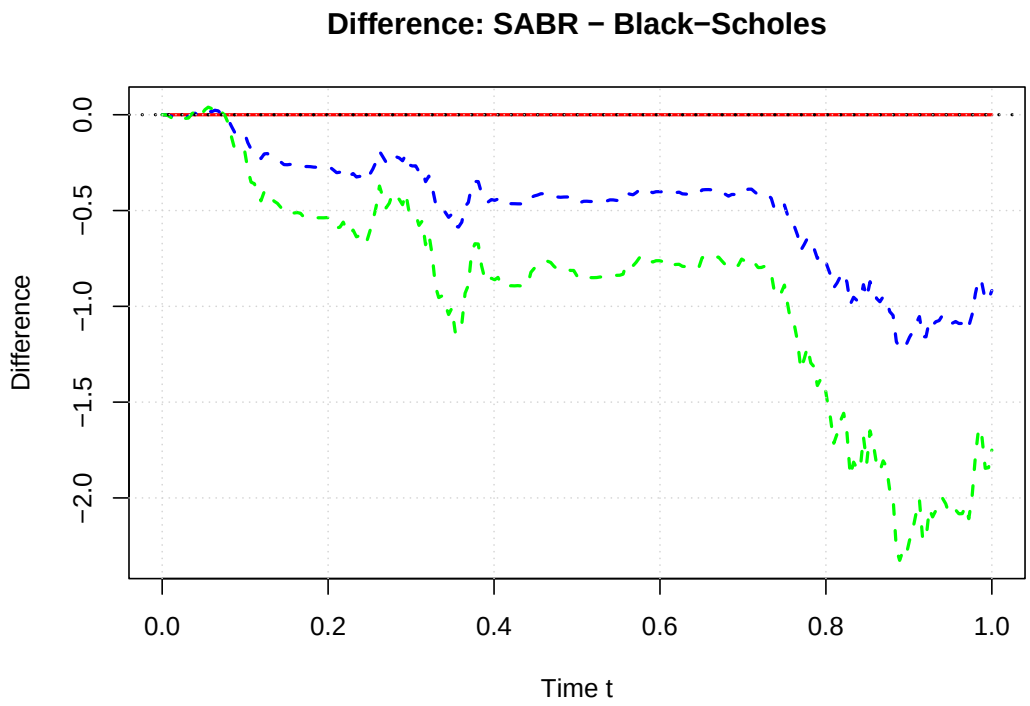
A crucial verification is to check that the SABR model reduces to the Black-Scholes model when $\beta = 1$ and $\nu = 0$. Furthermore, keeping beta =1, the further away v moves from zero the greater the difference in the price paths. We compare paths generated with the same random shock forthree different values of v.

```
> # Parameters
> S0 <- 100; mu <- 0.05; H  <- 1; n <- 252
> beta  <- 1; rho   <- -0.3; alpha0 <- 0.2
> # Step 1: Generate correlated shocks
> Shocks <- sabr.CorrShocks(rho, n, H)
> # Step 2:  Brownian/BS path with SAME Z1
> BS_price <- brownian.path(S0, mu, alpha0, n, H, Shocks$Z1)
> # Step 4: SABR price path using SAME Z1
> SABR_price_v0 <- sabr.path(S0, alpha0, mu, beta, 0, rho, n, H,
+                            Shocks$Z1, Shocks$Z2c)
> SABR_price_v10 <- sabr.path(S0, alpha0, mu, beta, 0.10, rho, n, H,
+                             Shocks$Z1, Shocks$Z2c)
> SABR_price_v20 <- sabr.path(S0, alpha0, mu, beta, 0.20, rho, n, H,
+                             Shocks$Z1, Shocks$Z2c)

> # Plot both paths
> tgrid <- seq(0, H, length.out = n + 1)
> plot(tgrid, BS_price, type = "l", lwd = 2, col = "blue",
+      main = "Black-Scholes vs SABR (beta=1, nu=0)",
+      xlab = "Time t", ylab = "S(t)")
```

```
> lines(tgrid, SABR_price_v0$S, lwd = 2, lty = 2, col = "red")
> lines(tgrid, SABR_price_v10$S, lwd = 2, lty = 2, col = "black")
> lines(tgrid, SABR_price_v20$S, lwd = 2, lty = 2, col = "green")
> legend("topleft", legend = c("Black-Scholes", "SABR (nu=0)",
+                              "SABR (nu=.10)", "SABR (nu=.20)"),
+        lty = c(1, 2, 3, 4), lwd = 2, col = c("blue", "red","black", "green"),
+        bty = "n")
> grid()
>
```




```
> # Plot the difference
> diff1 <- SABR_price_v0$S - BS_price
> diff2 <- SABR_price_v10$S - BS_price
> diff3 <- SABR_price_v20$S - BS_price
> ylim <- range(c(diff1, diff2, diff3))
> plot(tgrid, diff1, , ylim = ylim, type = "l", lwd = 2, col = "red",
+      main = "Difference: SABR - Black-Scholes",
+      xlab = "Time t", ylab = "Difference")
> lines(tgrid, diff2, lwd = 2, lty = 2, col = "blue")
> lines(tgrid, diff3, lwd = 2, lty = 2, col = "green")
> abline(h = 0, lty = 3, lwd = 2, col = "black")
> grid()
```



The first figure compares the Black-Scholes path with the SABR path under identical random shocks but with varying levels of v. Evidently, the two paths are identical for v $= 0$, confirming that our implementation correctly reduces to the Black-Scholes model in this special case.

The second figure shows the difference between the two paths for the various levels of v, indicating that the further v moves from zero the greater the difference in the price of the two models.

5.6 Experiment 6: Comparison with Black-Scholes (option price)

To check the correctness of the Monte Carlo implementation for pricing the European call, we set up the function parameters under the Black-Scholes condition ($\beta = 1$, $\nu = 0$, $\alpha_0 = \sigma$)

```
> S0  <- 120
> r   <- 0.02
> E   <- 100
> mu  <- 0.05
> H   <- 1
> n   <- 250
> beta<- 1
> v   <- 0
> rho <- 0
> alpha0 <- 0.2
> n_steps <- 250
> N <- 80000
> sigma <- 0.2
> sabr.europ.mc(S0, alpha0, E, r, beta, v, rho, H, n_steps, N)

$value
[1] 23.73999

$confint
[1] 23.67556 23.80442

$std_error
[1] 0.03287329
```

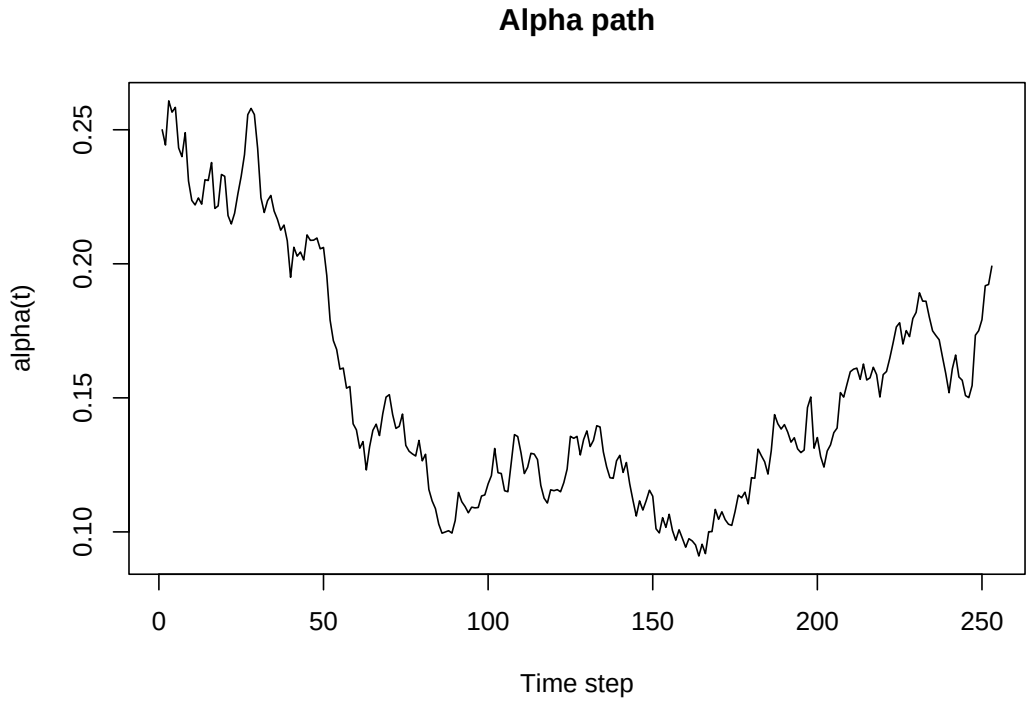
```
> bs.europ.formula(S0,E,r,sigma,H)

[1] 23.7421

>
>
```

We can say with 95 per cent confidence that the Black-Scholes price equals the SABR price under the Black-Scholes conditions.

5.7 Experiment 7: Value a risky out-of-the-money option

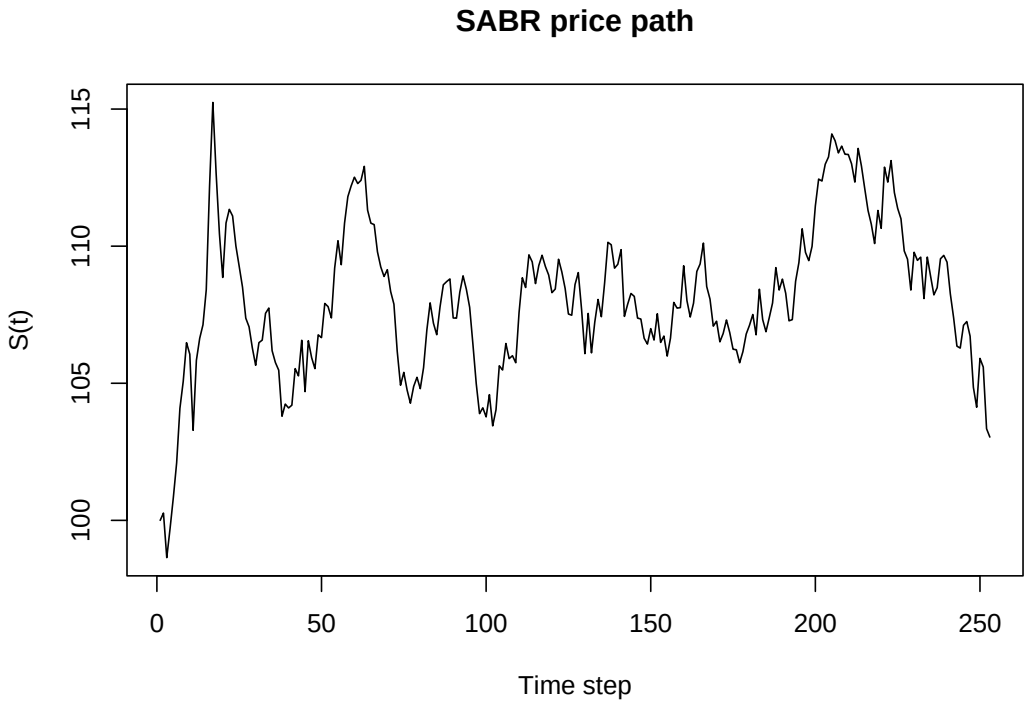
Here we will test our `sabr.europ.mc` function with high vol. of vol. and high vol. changes, negative correlation and large number of simulations.

```
> S0      <- 100
> E       <- 160
> r       <- 0.04
> mu      <- 0.04
> H       <- 1
> beta    <- 1
> v       <- 0.7
> rho     <- -0.4
> alpha0  <- 0.25
> n_steps <- 252
> N       <- 100000
> shocks <- sabr.CorrShocks( rho, n_steps, H)
> test_alpha_path <- sabr.AlphaPath(alpha0, v, n_steps, H, shocks$Z2c)
> plot(test_alpha_path, type = "l", main = "Alpha path", xlab = "Time step", ylab = "alpha(t)")
```



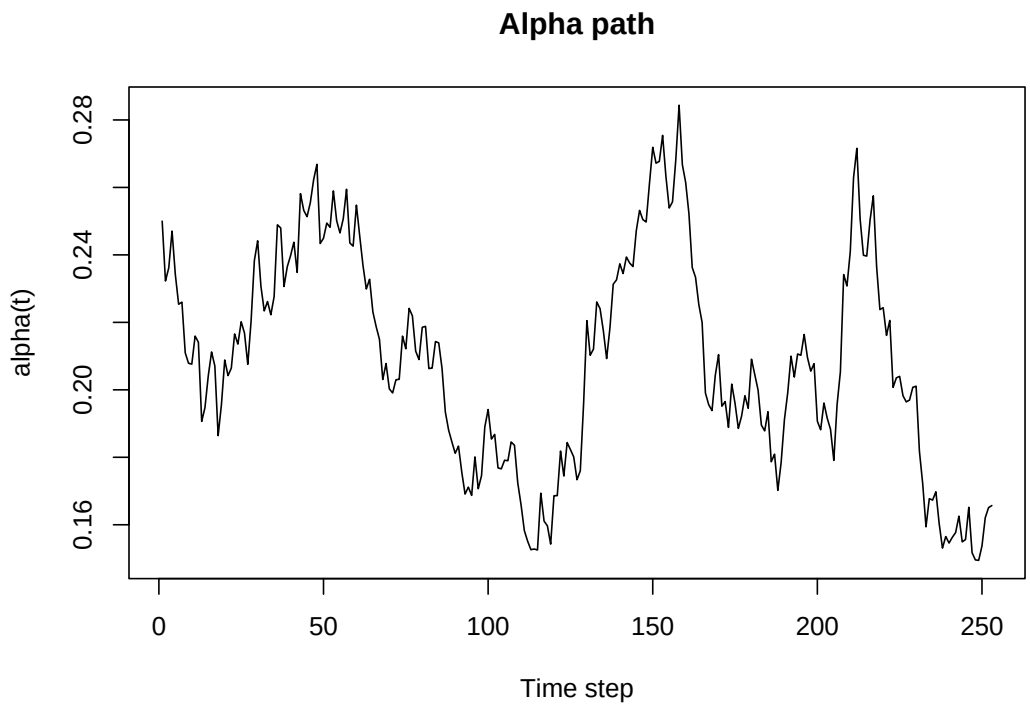

```
> test_sabr_path <- sabr.path(S0,alpha0,mu,beta,v,rho,n_steps,H, shocks$Z1,
+                             shocks$Z2c, return_alpha = F)
> plot(test_sabr_path, type = "l", main = "SABR price path", xlab = "Time step", ylab = "S(t)")
> sabr.europ.mc(S0, alpha0, E, r, beta, v, rho, H, n_steps, N)
```

```
$value
[1] 0.5188305

$confint
[1] 0.4866797 0.5509814

$std_error
[1] 0.01640349
```

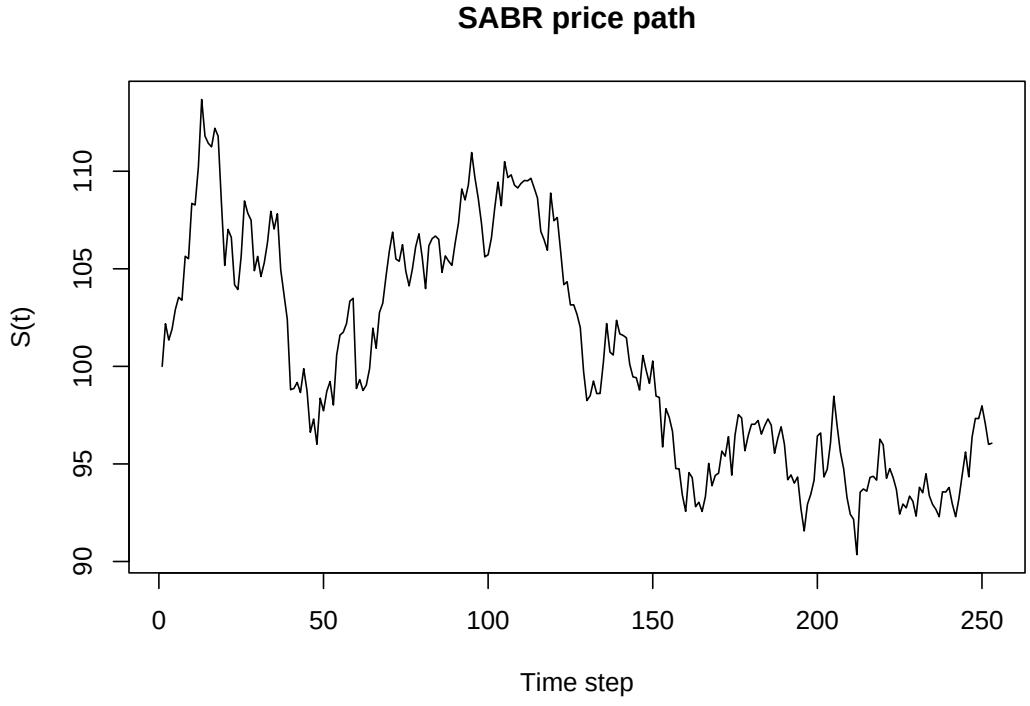


We will try again with a different seed.

```
> set.seed(42)
> shocks <- sabr.CorrShocks( rho, n_steps, H)
> test_alpha_path <- sabr.AlphaPath(alpha0, v, n_steps, H, shocks$Z2c)
> plot(test_alpha_path, type = "l", main = "Alpha path", xlab = "Time step", ylab = "alpha(t)")
```



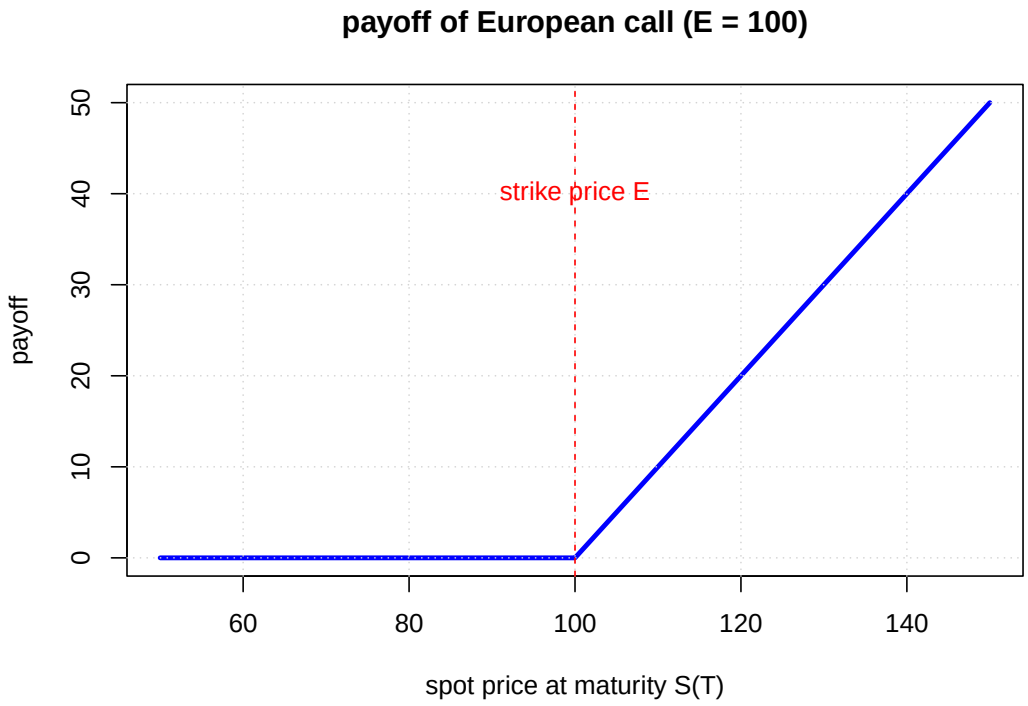

```
> test_sabr_path <- sabr.path(S0,alpha0,mu,beta,v,rho,n_steps,H, shocks$Z1, shocks$Z2c, return_alp
> plot(test_sabr_path, type = "l", main = "SABR price path", xlab = "Time step", ylab = "S(t)")
> sabr.europ.mc(S0, alpha0, E, r, beta, v, rho, H, n_steps, N)

$value
[1] 0.5102661

$confint
[1] 0.4773881 0.5431441

$std_error
[1] 0.01677449
```



The test confirms that the Monte Carlo estimator produces stable pricing results (≈ 0.51) with low standard error across independent runs, even for a highly volatile, deep out-of-the-money option.

5.8 Experiment 8: Payoff function for varying asset price at maturity

```
> set.seed(123)
> E <- 100
> # sequence of underlying prices at maturity
> S_T <- seq(from = 50, to = 150, by = 1)
> Payoff <- pmax(S_T - E, 0)
> plot(S_T, Payoff,
+      type = "l",
+      lwd = 3,
+      col = "blue",
+      main = "payoff of European call (E = 100)",
+      xlab = "spot price at maturity S(T)",
+      ylab = "payoff")
> # grid for better readability
> grid()
> abline(v = E, col = "red", lty = 2)
> text(E, 40, "strike price E", col = "red")
```

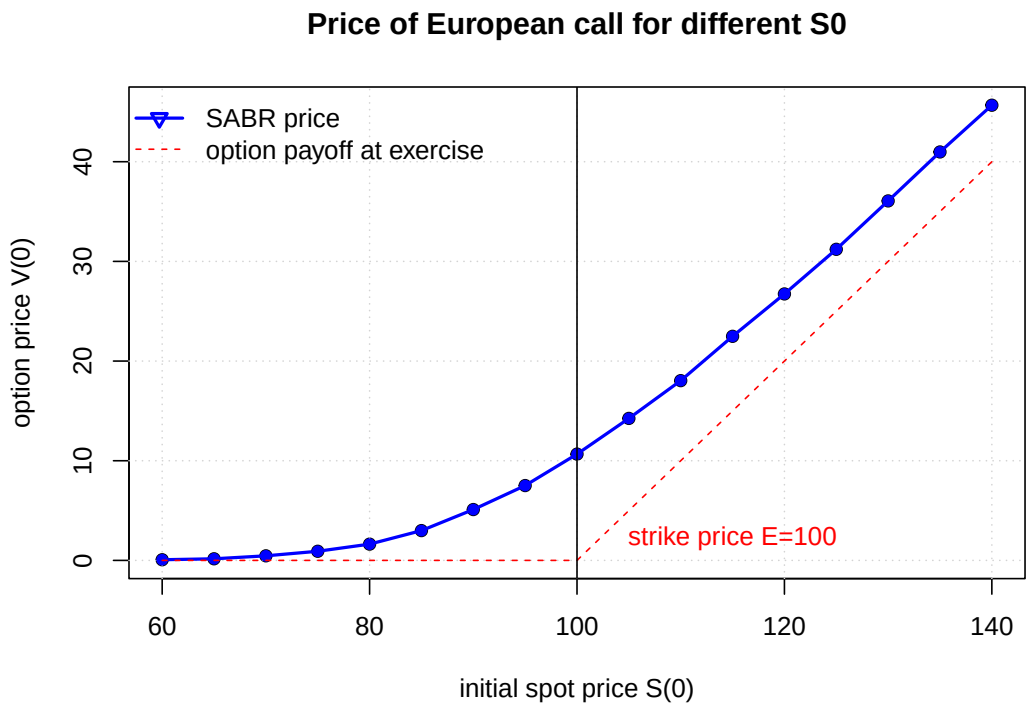


This plot shows the payoff function of the European call with a strike price $E = 100$. We can see that the upside potential for profit is unlimited, while the losses are restricted from below at 0.

5.9 Experiment 9: European call value for varying asset price at t=0

```
> set.seed(123)
> E       <- 100
> r       <- 0.05
> H       <- 1
> alpha0  <- 0.2
> beta    <- 1
> v       <- 0.5
> rho     <- -0.3
```

```
> n_steps <- 252
> N       <- 10000
> # range of initial spot prices S0
> S0_seq <- seq(from = 60, to = 140, by = 5)
> # storage vectors
> sabr_prices <- numeric(length(S0_seq)) #vector of prices for each sim.
> # loop to find SABR price for each S0 in S0_seq
> for(i in 1:length(S0_seq)) {
+   res <- sabr.europ.mc(S0 = S0_seq[i], alpha0, E, r, beta, v, rho, H, n_steps, N)
+
+   # store values and confidence intervals
+   sabr_prices[i] <- res$value
+
+ }
> plot(S0_seq, sabr_prices,
+      ylim = c(0, max(sabr_prices)),
+      main = "Price of European call for different S0",
+      xlab = "initial spot price S(0)",
+      ylab = "option price V(0)")
> grid()
> lines(S0_seq, sabr_prices, col = "blue", lwd = 2)
> points(S0_seq, sabr_prices, pch = 16, col = "blue")
> # add the S(T) against payoff function
> lines(S0_seq, pmax(S0_seq - E, 0), col = "red", lty = 2)
> legend("topleft",
+        legend = c("SABR price",
+        "option payoff at exercise"),
+        col = c("blue", "red"),
+        lty = c(1, 2),
+        lwd = c(2, 1),
+        pch = c(6, NA),
+        bty = "n"
+        )
> text(115, 0, "strike price E=100", pos=3, col="red")
> abline(v = E, col = "black", lty = 1)
```

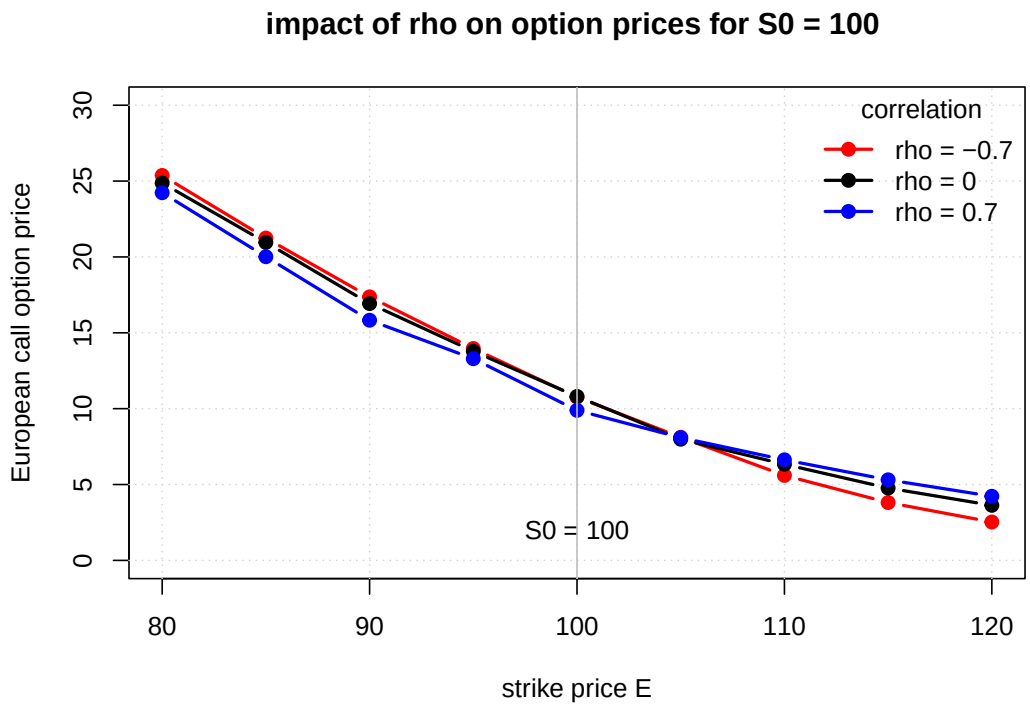


The plot compares the European call option prices calculated using the SABR model against the option's payoff for varying spot prices (S_0), given a strike price of 100. It shows that the option price consistently remains above the intrinsic payoff due to time value. The risk premium is the highest at the money ($S_0 \approx 100$).

5.10 Experiment 10: effect of correlation on price

The plot illustrates how the correlation ρ affects the pricing of European call options for $S_0 = 100$ over different strike prices E.

```
> S0 <- 100
> E_seq   <- seq(80, 120, by=5) # range of strike prices
> rhos    <- c(-0.7, 0, 0.7)
> colors  <- c("red", "black", "blue")
> # setup plot
> plot(NULL, xlim=range(E_seq), ylim=c(0, 30),
+      main="impact of rho on option prices for S0 = 100 ",
+      xlab="strike price E", ylab="European call option price")
> grid()
> for(i in 1:length(rhos)) {
+
+   price_vector <- sapply(E_seq, function(strike) {
+     sabr.europ.mc(S0=100, alpha0=0.2, E=strike, r=0.05,
+                   beta=1, v=0.5, rho=rhos[i], H=1, n_steps=252, N=5000)$value
+   })
+
+   lines(E_seq, price_vector, col=colors[i], lwd=2, type="b", pch=19)
+ }
> legend("topright",
+        legend=paste("rho =", rhos),
+        col=colors, lwd=2, pch=19, title="correlation", bty="n")
> text(100,2, "S0 = 100")
> abline(v=100,lty = 1, col = "grey")
>
>
```



When ρ is positive, an increase in the asset price is associated with an increase in volatility. Out-of-the-money calls with high strikes become more expensive because there is a higher probability of the asset reaching extreme high prices.

When ρ is negative, a decrease in asset price leads to higher volatility, which is standard in equity markets. Deep in-the-money calls become more expensive because the option price reflects the higher probability of extreme downside moves.

6 SABR Calibration

6.1 Introduction

Implied volatility varies across strikes and maturities. SABR model calibration is the process of choosing parameters (α, ρ, ν) (with β often fixed) so that the model's implied Black volatilities $\sigma_{\text{SABR}}(K, T)$ match the market-implied volatilities observed for a given expiry T. In practice, markets quote option prices and, equivalently, implied volatilities. In FX, for example, they are commonly quoted as Black (Garman-Kohlhagen) implied vols using standard smile quotes for each expiry T.

Rather than pricing each strike via Monte Carlo simulation, practitioners typically use the closed-form SABR implied volatility approximation introduced by Hagan et al., which provides $\sigma_{\text{SABR}}(K, T)$ as a function of the strike K, the forward F, and the SABR parameters:

$$\sigma_{\text{SABR}}(F, K, T, \alpha, \beta, \rho, \nu) = \frac{\alpha}{\mathcal{D}(F, K)} \frac{z}{x(z)} \left[1 + A(F, K) \cdot T\right] \tag{19}$$

where

$$\mathcal{D}(F, K, \beta) := (FK)^{\frac{1-\beta}{2}} \left[1 + \frac{(1-\beta)^2}{24} \ln^2\left(\frac{F}{K}\right) + \frac{(1-\beta)^4}{1920} \ln^4\left(\frac{F}{K}\right)\right],$$

$$z(F, K, \beta, \nu) := \frac{\nu}{\alpha} (FK)^{\frac{1-\beta}{2}} \ln\left(\frac{F}{K}\right),$$

$$x(z, \rho) := \ln\left(\frac{\sqrt{1 - 2\rho z + z^2} + z - \rho}{1 - \rho}\right),$$

$$A(F, K, \alpha, \beta, \rho, \nu) := \frac{(1-\beta)^2}{24} \frac{\alpha^2}{(FK)^{1-\beta}} + \frac{\rho\, \beta\, \nu}{4} \frac{\alpha}{(FK)^{\frac{1-\beta}{2}}} + \frac{2 - 3\rho^2}{24} \nu^2.$$

In the special case, where the option is at-the-money, i.e. the strike price K is equal to the forward rate T, the formula simplifies to

$$\sigma_{\text{ATM}}(F,K,T,\alpha,\beta,\rho,\nu) = \frac{\alpha}{F^{1-\beta}}\left[1+T\left(\frac{(1-\beta)^2}{24}\frac{\alpha^2}{F^{2-2\beta}}+\frac{\rho\,\beta\,\nu}{4}\frac{\alpha}{F^{1-\beta}}+\frac{2-3\rho^2}{24}\nu^2\right)\right]. \tag{20}$$

There are multiple calibration methods. It is possible to calibrate Alpha, Rho and Nu directly, but in our example we will imply Alpha from the at-the-money volatility. To do so we start with the special case ATM approximation and we solve it for Alpha.

$$\begin{aligned}
\sigma_{\text{ATM}}^{\text{mkt}} &= \frac{\alpha}{F^{1-\beta}}\left[1+T\left(\frac{(1-\beta)^2}{24}\frac{\alpha^2}{F^{2-2\beta}}+\frac{\rho\,\beta\,\nu}{4}\frac{\alpha}{F^{1-\beta}}+\frac{2-3\rho^2}{24}\nu^2\right)\right] \\
\iff \qquad 0 &= \left(\frac{(1-\beta)^2\,T}{24\,F^{2-2\beta}}\right)\alpha^3+\left(\frac{\rho\,\beta\,\nu\,T}{4\,F^{1-\beta}}\right)\alpha^2+\left(1+\frac{(2-3\rho^2)\,\nu^2\,T}{24}\right)\alpha-\sigma_{\text{ATM}}^{\text{mkt}}\,F^{1-\beta}.
\end{aligned}$$

where $\nu \geq 0$, $\rho \in (-1,1)$, $\beta \in [0,1]$, $F>0$ and $T>0$.

We end up with a cubic that may have multiple real and complex roots. One would normally choose a positive root, since $\alpha > 0$ is a volatility level. If there are multiple positive roots, one would usually pick the one that is economically sensible or continuous across nearby expiries. In our case, later we will pick the smallest positive real root.

6.2 Calibration algorithm

1. **Collect and preprocess market quotes (for a fixed expiry T).**
   Obtain market-implied vols for several strikes $\left(K_i, \sigma_i^{\text{mkt}}\right)$. Compute the forward F for that expiry and use the correct year fraction.

2. **Fix** β (choose based on market convention or product).

3. **For a candidate (ρ^*,ν^*), determine α from the ATM condition.**
   Solve the ATM implied-volatility equation for α such that the model ATM implied vol matches the market ATM implied vol:

$$\sigma_{\text{ATM}}^{\text{SABR}}\big(F,T;\alpha=g(\rho^*,\nu^*),\beta,\rho^*,\nu^*\big)=\sigma_{\text{ATM}}^{\text{mkt}}.$$

4. **Compute model vols at all strikes.**
   Use Hagan's approximation to compute

$$\sigma_i^{\text{SABR}}=\sigma_{\text{Hagan}}\big(F,K_i,T;\alpha=g(\rho^*,\nu^*),\beta,\rho^*,\nu^*\big), \qquad i=1,\ldots,N.$$

5. **Calculate the error.**
   For the candidate pair (ρ^*,ν^*) and strike K_i, define the residual as

$$r_i(\rho^*,\nu^*)=\sigma_i^{\text{mkt}}-\sigma_i^{\text{SABR}}(\rho^*,\nu^*), \qquad i=1,\ldots,N.$$

6. **Optimize (ρ,ν).**
   Define the objective function as the sum of squared errors:

$$\text{SSE}(\rho^*,\nu^*)=\sum_{i=1}^{N} r_i(\rho^*,\nu^*)^2.$$

   Choose $(\hat{\rho},\hat{\nu})$ that minimizes the SSE:

$$(\hat{\rho},\hat{\nu})=\arg\min_{\rho\in(-1,1),\,\nu>0}\ \text{SSE}(\rho^*,\nu^*).$$

6.3 Implementation in R

First we will define the Hagan's closed-form SABR implied volatility approximation function and its special-case expression, when the option is at-the-money, which is used for numerical stability when the forward rate is approximately equal to the strike price:

```
> # ATM SABR implied volatility approximation (special-case Hagan formula)
> #
> # Purpose: Calculate the model implied volatility of a ATM option using
> #          the special-case formula
> #
> # Inputs:
> #  --- F_0:   forward price/rate (> 0); numeric scalar or vector
> #  --- H:     time to expiry in years (> 0); numeric scalar
> #  --- alpha: SABR volatility level (> 0); numeric scalar
> #  --- beta:  SABR elasticity (typically in [0, 1]); numeric scalar
> #  --- rho:   correlation parameter with abs(rho) < 1; numeric scalar
> #  --- v:     vol-of-vol (>= 0); numeric scalar
> #
> # Output:
> #  --- vol_atm: implied Black volatility at-the-money; numeric vector
>
> sabr_atm_vol <- function(F_0, H, alpha, beta, rho, v) {
+   alpha / (F_0^(1 - beta)) *
+     (1 + H * (
+       ((1 - beta)^2 / 24) * (alpha^2 / (F_0^(2 - 2*beta))) +
+         ((rho * beta * v) / 4) * (alpha / (F_0^(1 - beta))) +
+         ((2 - 3 * rho^2) / 24) * (v^2)
+     )
+     )
+ }

> # SABR implied volatility approximation (general Hagan formula)
> #
> # Purpose: Computes SABR model implied volatility
> #
> # Inputs:
> #  --- F_0:   forward price/rate (F_0 > 0); numeric scalar or vector
> #  --- K:     strike (K > 0); numeric scalar or vector
> #  --- H:     time to expiry in years (H > 0); numeric scalar
> #  --- alpha: SABR volatility level (alpha > 0); numeric scalar
> #  --- beta:  SABR elasticity (typically in [0, 1]); numeric scalar
> #  --- rho:   correlation parameter with abs(rho) < 1; numeric scalar
> #  --- v:     vol-of-vol (v >= 0); numeric scalar
> #
> # Output:
> #  --- vol: implied Black volatility for each (F_0, K) pair; numeric vector
> #
> # Notes:
> #  --- Uses the function sabr_atm_vol(F_0, H, alpha, beta, rho, v)
> #      for the ATM case
>
> sabr_vol <- function(F_0, K, H, alpha, beta, rho, v) {
+
+   if (any(F_0 <= 0) || any(K <= 0)) stop("F_0 and K must be > 0
+                                          (Black/lognormal SABR).")
+   if (alpha <= 0)                   stop("alpha must be > 0")
```

```
+   if (H <= 0)                    stop("H must be > 0")
+   if (abs(rho) >= 1)             stop("|rho| must be < 1")
+   if (v < 0)                    stop("v must be >= 0")
+
+   # Define variables for repeatedly appearing calculations
+   one_m_b <- 1 - beta
+   logFK   <- log(F_0 / K)    # log-moneyness
+   abslog  <- abs(logFK)
+   FK_beta <- (F_0 * K)^(one_m_b / 2)
+
+   # Denominator correction - adjusts for log(F_0/K) terms
+   log2 <- logFK^2
+   log4 <- log2^2
+   denom <- FK_beta * (1 + (one_m_b^2/24) * log2 + (one_m_b^4/1920) * log4)
+
+   # Hagan variables z and x(z)
+   z  <- (v / alpha) * FK_beta * logFK
+   xz <- log((sqrt(1 - 2*rho*z + z^2) + z - rho) / (1 - rho))
+
+   # z/x(z) has a removable singularity at z=0 ATM
+   # Use the limit z/x(z) -> 1 for numerical stability
+   z_over_x <- ifelse(abs(z) < 1e-08, 1, z / xz)
+
+   # Time correction term - accounts for expiry horizon H
+   term1 <- (one_m_b^2/24) * (alpha^2 / ((F_0*K)^(one_m_b)))
+   term2 <- (rho*beta*v/4) * (alpha / FK_beta)
+   term3 <- ((2 - 3*rho^2) * v^2 / 24)
+
+   vol <- (alpha / denom) * z_over_x * (1 + (term1 + term2 + term3) * H)
+
+   # Near ATM situation -> use explicit ATM formula when log(F_0/K) is very small
+   idx_atm <- abslog < 1e-10
+   if (any(idx_atm)) vol[idx_atm] <- sabr_atm_vol(F_0, H, alpha, beta, rho, v)
+
+   vol
+ }
```

Next, we will define the R functions used for the numerical rootfinding of α from the special-case ATM implied volatility approximation equation.

```
> # Numerical rootfinding of alpha from ATM via the cubic (Helper)
> #
> # Purpose: If we rearrange the ATM approximation equation to solve for alpha,
> #          we obtain a polynomial equation in alpha. In this implementation the
> #          rearrangement is written as a cubic:
> #               a3 * alpha^3 + a2 * alpha^2 + a1 * alpha + a0 = 0
> #
> #          This function constructs the cubic coefficients (a0..a3) implied by
> #          the ATM approximation and returns *all* roots of the polynomial
> #          using polyroot().
> #
> # Inputs:
> #  --- rho:       SABR correlation (abs(rho) < 1)
> #  --- v:         vol-of-vol (>= 0)
> #  --- F_0:       forward price/rate (> 0)
> #  --- H:         time to expiry in years (> 0)
```

```
> #  --- beta:      elasticity (typically in [0, 1])
> #  --- sigma_atm: observed market ATM implied Black volatility (> 0)
> #
> # Output:
> #  --- roots: complex vector of length 3 containing the cubic roots for alpha
> #
> # Notes:
> #  --- This is a numerical helper, real_alpha_roots() is used to pick alpha
> #  --- polyroot() can return complex roots even when real solutions exist
>
> alpha_roots <- function(rho, v, F_0, H, beta, sigma_atm) {
+
+   # a3*alpha^3 + a2*alpha^2 + a1*alpha + a0 = 0
+   a3 <- (1 - beta)^2 * H / (24 * F_0^(2 - 2*beta))
+   a2 <- rho * beta * v * H / (4 * F_0^(1 - beta))
+   a1 <- 1 + (2 - 3 * rho^2) * v^2 * H / 24
+   a0 <- -sigma_atm * F_0^(1 - beta)
+
+   # Find the roots
+   polyroot(c(a0, a1, a2, a3))
+ }

> # Numerical finding of real roots from ATM via the cubic
> #
> # Purpose: Calls alpha_roots(...) to compute all cubic roots for alpha.
> #          Then it filters those roots to keep only roots that are (numerically)
> #          real (imaginary part close to zero) and positive. Finally, it returns
> #          the smallest positive real root
> #
> # Inputs:
> #  --- rho:       SABR correlation (|rho| < 1); controls skew/asymmetry
> #  --- v:         vol-of-vol (>= 0); controls smile curvature
> #  --- F_0:       forward price/rate (> 0)
> #  --- H:         time to expiry in years (> 0)
> #  --- beta:      elasticity (typically in [0, 1])
> #  --- sigma_atm: observed market ATM implied Black volatility (> 0)
> #  --- imag_tol:  tolerance for treating complex roots as real (default 1e-6)
> #
> # Output:
> #  --- alpha_hat: calibrated alpha implied by ATM, chosen as the smallest
> #                 positive real root of the cubic
> #
> # Notes:
> #  --- Stops with an error if no positive real root is found (handled later
> #      in the full calibration routine via tryCatch(...) block)
> #  --- Numerical root solvers typically return tiny imaginary parts due to
> #      floating point error (e.g. 0.5000000 + 1e-12 i). We treat such roots as
> #      real if the imaginary part is below a tolerance parameter imag_tol
>
> real_alpha_roots <- function(rho, v, F_0, H, beta, sigma_atm, imag_tol = 1e-6) {
+   # Find all roots
+   roots <- alpha_roots(rho, v, F_0, H, beta, sigma_atm)
+
+   # Take only real positive roots
+   real_roots          <- Re(roots)[abs(Im(roots)) < imag_tol]
+   positive_real_roots <- real_roots[real_roots > 0]
```

```
+
+   if (length(positive_real_roots) == 0) stop("No positive real root for alpha.")
+
+   # Return the smallest root
+   min(positive_real_roots)
+ }
```

Additionally, we will introduce a small helper function that calculates the year fraction between two dates. We assume that the year has 365.242199 days.

```
> # Year fraction between two dates
> #
> # Purpose: Convert two calendar dates into a time-to-expiry expressed in years.
> #          This is needed in SABR because the model formulas use a continuous
> #          time horizon H in years
> #
> # Inputs:
> #  --- SettleDate:   valuation/settlement date (Date / POSIXct)
> #  --- ExerciseDate: option expiry/exercise date (Date / POSIXct)
> #
> # Output:
> #  --- H: time difference in years (numeric)
> #
> # Notes:
> #  --- Uses 365.242199 days per year (mean tropical year) as a constant scaling
>
> yearfrac <- function(SettleDate, ExerciseDate) {
+   as.numeric(difftime(ExerciseDate, SettleDate, units = "days")) / 365.242199
+ }
```

After defining the helper functions, we implement the full calibration routines. We start with a single-expiry calibration: for a given maturity we infer Alpha from the ATM quote and then fit Rho and Nu by minimizing the squared error between market and SABR-implied Black volatilities across strikes. We then apply this single-expiry routine iteratively across all expiries to obtain a complete set of SABR parameters and construct an implied volatility surface over strikes and maturities.

```
> # Calibration of SABR parameters (rho, v) for ONE expiry by using implied
> # alpha from ATM
> #
> # Purpose: Calibrate the SABR smile for a single maturity H by:
> #          (1) identifying the ATM quote (closest strike to F_0)
> #          (2) solving for alpha from the ATM volatility (via real_alpha_roots),
> #              given candidate (rho*, v*) and fixed beta
> #          (3) fitting (hat_rho, hat_nu) by minimizing the sum of squared errors
> #              between market implied vols and SABR model vols across strikes
> #          (4) recomputing the final alpha implied by ATM using the fitted
> #              (hat_rho, hat_nu).
> #
> # Inputs:
> #  --- SettleDate:          settlement date (used in yearfrac)
> #  --- ExerciseDate:        option expiry/exercise date (used in yearfrac)
> #  --- F_0:                 forward price/rate at SettleDate (> 0)
> #  --- K_vec:               vector of strikes (> 0), same length as vol_vec
> #  --- vol_vec:             vector of market implied Black vols, same length as
> #                           K_vec
> #  --- beta:                fixed SABR elasticity (usually in [0, 1], often 0.5)
> #  --- start:               initial guess for optimizer parameters (rho, v)
```

```
> #  --- lower:               lower bounds for (rho, v) in optimization
> #  --- upper:               upper bounds for (rho, v) in optimization
> #  --- include_atm_in_fit:  if FALSE (default), sets the ATM residual to 0 so ATM
> #                           is not overweighted (alpha is already pinned to ATM)
> #
> # Output (list):
> #  --- Alpha:    final alpha implied from ATM using calibrated (hat_rho, hat_nu)
> #  --- Beta:     beta used (fixed)
> #  --- Rho:      calibrated rho (smile skew/asymmetry)
> #  --- Nu:       calibrated v (smile curvature / vol-of-vol)
> #  --- SSE:      final sum of squared errors across strikes
> #  --- conv:     optim() convergence code; 0 typically indicates success
> #  --- idx_atm:  index of the strike treated as ATM (closest to F_0)
> #
> # Notes:
> #  --- Uses yearfrac(SettleDate, ExerciseDate) to compute H in years
> #  --- Uses optim(..., method="L-BFGS-B") to fit (rho, v) with bounds
> #  --- Uses penalty residuals (1e3) to keep optimizer away from invalid regions
> #      (abs(rho) close to 1, v < 0 or when alpha cannot be solved robustly)
> #  --- Requires sabr_vol(), real_alpha_roots() and their dependencies.
>
> calibrate_sabr_one_expiry <- function(SettleDate, ExerciseDate, F_0, K_vec, vol_vec,
+                                       beta = 0.5,
+                                       start = c(0, 0.5),
+                                       lower = c(-0.999, 1e-8),
+                                       upper = c( 0.999, 10.0),
+                                       include_atm_in_fit = FALSE) {
+
+   # Identify which strike is ATM (closest to F_0)
+   idx_atm <- which.min(abs(K_vec - F_0))
+   sigma_atm <- vol_vec[idx_atm]
+
+   # Convert dates to year fraction
+   H <- yearfrac(SettleDate, ExerciseDate)
+
+   # Residuals vector (MarketVols - ModelVols)
+   residual_vec <- function(X) {
+     rho <- X[1]
+     v  <- X[2]
+
+     # Boundary penalties - keep optimizer away from invalid regions
+     if (abs(rho) > 0.9999 || v < 0) return(rep(1e3, length(K_vec)))
+
+     # Robust calculation of the real positive roots of alpha
+     alpha <- tryCatch(
+       real_alpha_roots(rho, v, F_0, H, beta, sigma_atm),
+       error = function(e) NA_real_
+     )
+     # Penalize when alpha cannot be solved robustly
+     if (!is.finite(alpha) || alpha <= 0) return(rep(1e3, length(K_vec)))
+
+     # Compute model vols across strikes using Hagan SABR
+     model <- sabr_vol(F_0, K_vec, H, alpha, beta, rho, v)
+
+     # Residuals (vector)
+     res <- vol_vec - model
```

```
+
+     # (OPTIONAL) Remove ATM from fitting because alpha is forced to match ATM to
+     # prevent giving the ATM extra influence relative to the other strikes
+     if (!include_atm_in_fit) res[idx_atm] <- 0
+
+     res
+   }
+
+   # Least-squares objective
+   # minimize sum(residuals^2)
+   sse_obj <- function(X) {
+     r <- residual_vec(X)
+     sum(r^2)
+   }
+
+   # Optimization over (rho, v) with bounds
+   fit <- optim(
+     par    = start,
+     fn     = sse_obj,
+     method = "L-BFGS-B",
+     lower  = lower,
+     upper  = upper
+   )
+
+   # Extract calibrated parameters
+   rho_hat <- fit$par[1]
+   nu_hat  <- fit$par[2]
+
+   # Final alpha implied from ATM using calibrated (rho_hat, nu_hat)
+   alpha_hat <- real_alpha_roots(rho_hat, nu_hat, F_0, H, beta, sigma_atm)
+
+   list(
+     Alpha   = alpha_hat,
+     Beta    = beta,
+     Rho     = rho_hat,
+     Nu      = nu_hat,
+     SSE     = fit$value,
+     conv    = fit$convergence,
+     idx_atm = idx_atm
+   )
+ }

> # Calibration of a SABR volatility SURFACE across multiple expiries
> #
> # Purpose: Build a SABR parameter surface by calibrating SABR independently for
> #          each expiry (each column of the market smile data). For every expiry:
> #            (1) pick the forward F_0 for that maturity
> #            (2) take the vector of strikes K and market vols for that maturity
> #            (3) call calibrate_sabr_one_expiry(...) to fit (rho, v) and imply
> #                alpha from the ATM quote (given fixed beta)
> #          The results are then collected into a parameter table (one row per
> #          expiry)
> #
> # Inputs:
> #  --- SettleDate:           settlement date (used in yearfrac)
> #  --- ExerciseDates:        vector of expiries (length = n_exp)
```

```
> #  --- MarketStrikes:        matrix of strikes with dimensions [n_strikes x n_exp]
> #                           where column j corresponds to expiry ExerciseDates[j]
> #  --- MarketVolatilities:   matrix of market implied Black vols with the same
> #                           dimensions as MarketStrikes
> #  --- CurrentForwardValues: vector of forwards (length = n_exp), one forward per
> #                           expiry
> #  --- beta:                 fixed SABR elasticity (typically in [0, 1], often 0.5)
> #  --- start:                initial guess for (rho, v) for each expiry calibration
> #  --- include_atm_in_fit:   passed through to calibrate_sabr_one_expiry(); if FALSE
> #                           (default), ATM residual is set to 0 to avoid overweighting
> #
> # Output (list):
> #  --- params:  data.frame with one row per expiry containing:
> #               Alpha, Beta, Rho, Nu, SSE, conv
> #  --- details: list of length n_exp with the full calibration output for each expiry
> #               (including idx_atm and any other diagnostics returned)
> #
> # Notes:
> #  --- This is a per-expiry calibration, i.e. no smoothing across maturities
> #  --- Assumes MarketStrikes and MarketVolatilities are aligned column-wise to
> #      ExerciseDates and CurrentForwardValues
> #  --- Requires calibrate_sabr_one_expiry() and all underlying helpers
>
> calibrate_sabr_surface <- function(SettleDate, ExerciseDates,
+                                    MarketStrikes, MarketVolatilities,
+                                    CurrentForwardValues,
+                                    beta = 0.5,
+                                    start = c(0, 0.5),
+                                    include_atm_in_fit = FALSE) {
+
+   n_exp <- length(ExerciseDates)
+   res_list <- vector("list", n_exp)
+
+   # Iterate over each expiry to obtain the surface
+   for (j in 1:n_exp) {
+     res_list[[j]] <- calibrate_sabr_one_expiry(
+       SettleDate         = SettleDate,
+       ExerciseDate       = ExerciseDates[j],
+       F_0                  = as.numeric(CurrentForwardValues[j]),
+       K_vec              = as.numeric(MarketStrikes[, j]),
+       vol_vec            = as.numeric(MarketVolatilities[, j]),
+       beta               = beta,
+       start              = start,
+       include_atm_in_fit = include_atm_in_fit
+     )
+   }
+
+   # Pack results into a convenient table
+   out <- data.frame(
+     Alpha = sapply(res_list, function(x) x[["Alpha"]]),
+     Beta  = sapply(res_list, function(x) x[["Beta"]]),
+     Rho   = sapply(res_list, function(x) x[["Rho"]]),
+     Nu    = sapply(res_list, function(x) x[["Nu"]]),
+     SSE   = sapply(res_list, function(x) x[["SSE"]]),
+     conv  = sapply(res_list, function(x) x[["conv"]])
+   )
```

```
+
+   list(params = out, details = res_list)
+ }
```

Finally, we define two plotting functions: one that compares the market implied Black volatility smile to the SABR fitted smile for a chosen expiry and one that visualizes the calibrated SABR implied volatility surface across maturities and strikes.

```
> # Plot market volatility smile vs SABR fitted smile for one expiry
> #
> # Purpose: Visualize how well the calibrated SABR parameters match the observed
> #          market implied Black volatilities for a single expiry index j.
> #          The function takes the j-th maturity slice, computes the SABR fitted
> #          volatilities using the calibrated parameters and overlays the market
> #          smile (points) and SABR fitted smile (line)
> #
> # Inputs:
> #  --- j:                    integer index of the expiry slice to plot
> #  --- results:              calibration output from calibrate_sabr_surface()
> #                            (must contain results$details[[j]]
> #                            with Alpha/Beta/Rho/Nu)
> #  --- Settle:               settlement/valuation date (used in yearfrac)
> #  --- ExerciseDates:        vector of expiries (length = n_exp)
> #  --- MarketStrikes:        matrix [n_strikes x n_exp] of strikes
> #                            (column j = expiry j)
> #  --- MarketVolatilities:   matrix [n_strikes x n_exp] of market implied Black
> #                            vols aligned with MarketStrikes
> #  --- CurrentForwardValues: vector of forwards (length = n_exp), one per expiry
> #
> # Output:
> #  --- A base R plot
>
> plot_smile <- function(j, results, Settle, ExerciseDates, MarketStrikes,
+                        MarketVolatilities, CurrentForwardValues) {
+
+   pars    <- results$details[[j]]
+
+   F_0     <- as.numeric(CurrentForwardValues[j])
+   K       <- as.numeric(MarketStrikes[, j])
+   vol_mkt <- as.numeric(MarketVolatilities[, j])
+   H       <- yearfrac(Settle, ExerciseDates[j])
+
+   vol_fit <- sabr_vol(F_0, K, H, pars$Alpha, pars$Beta, pars$Rho, pars$Nu)
+
+   plot(K, vol_mkt, pch = 16, xlab = "Strike", ylab = "Black vol",
+        main = paste0(Maturities[j]," into ",Tenor, " Volatility Smile"))
+
+   lines(K, vol_fit, lwd = 2)
+
+   legend("topright", legend = c("Market", "SABR fit"),
+          pch = c(16, NA), lwd = c(NA, 2), bty = "n")
+ }
> # Plot SABR volatility surface
> #
> # Purpose:
> #   Plot the calibrated SABR implied volatility surface as a 3D surface using
```

```
> #   base R's persp(). The surface is evaluated on a common strike grid across
> #   all expiries, while maturities are represented by year fractions H
> #
> # What it does:
> #   1) Builds H_vec = yearfrac(SettleDate, ExerciseDates) (x-axis)
> #   2) Builds a common strike grid K_grid (y-axis)
> #   3) For each expiry j, pulls SABR params (Alpha, Beta, Rho, Nu) from fit$details[[j]]
> #      and computes vols on K_grid using sabr_vol(F0, K_grid, H_j, ...)
> #   4) Assembles the z-matrix for persp() with orientation:
> #        x = H_vec  (length nH)
> #        y = K_grid (length nK)
> #        z must be a matrix with nrow(z) = nH and ncol(z) = nK
> #      Therefore we compute vol_mat as [nK x nH] and pass z = t(vol_mat).
> #
> # Inputs:
> #  --- results:              calibration output from calibrate_sabr_surface()
> #                            (must contain results$details list)
> #  --- SettleDate:           Date object used to compute maturities in years
> #  --- ExerciseDates:        Date object used to compute maturities in years
> #  --- MarketStrikes:        strike matrix [n_strikes x n_exp]
> #                            (used only to set grid range)
> #  --- CurrentForwardValues: vector of forwards per expiry (length n_exp)
> #  --- strike_grid:          optional numeric vector;
> #                            if NULL, a grid is created automatically
> #  --- n_strike:             number of strike points if strike_grid is NULL
> #  --- theta, phi, expand, ticktype: passed to persp() for viewing angle/scale
> #
> # Output:
> #   - Draws a 3D surface plot
>
> plot_surface <- function(results,
+                          SettleDate, ExerciseDates,
+                          MarketStrikes, CurrentForwardValues,
+                          strike_grid = NULL,
+                          n_strike = 60,
+                          theta = 45, phi = 25, expand = 0.7,
+                          ticktype = "detailed") {
+
+   n_exp <- length(ExerciseDates)
+   if (n_exp != ncol(MarketStrikes))
+                     stop("ExerciseDates length must match ncol(MarketStrikes).")
+
+   if (n_exp != length(CurrentForwardValues))
+                     stop("CurrentForwardValues length must match ExerciseDates.")
+
+   if (is.null(results$details) || length(results$details) != n_exp)
+                     stop("results must be output of calibrate_sabr_surface().")
+
+   # x-axis: maturities in years
+   H_vec <- yearfrac(SettleDate, ExerciseDates)
+
+   # y-axis: strike grid
+   if (is.null(strike_grid)) {
+     K_min <- min(MarketStrikes, na.rm = TRUE)
+     K_max <- max(MarketStrikes, na.rm = TRUE)
+     K_grid <- seq(K_min, K_max, length.out = n_strike)
```

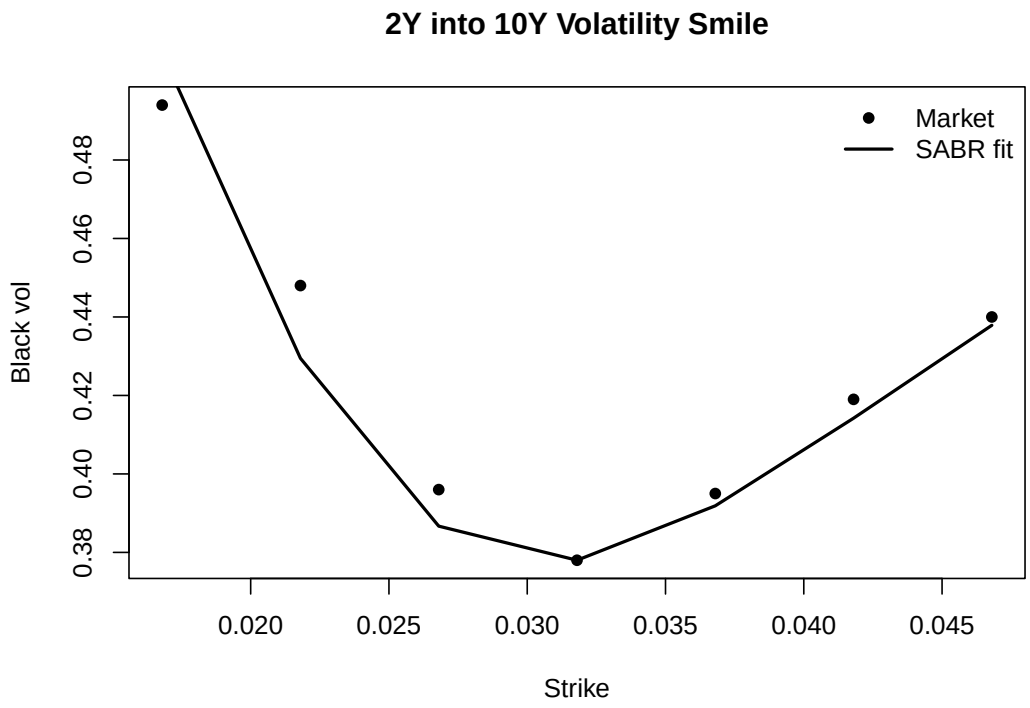
```
+   } else {
+     K_grid <- as.numeric(strike_grid)
+   }
+
+   # Compute vols on grid: vol_mat_KH is [K x H]
+   vol_mat_KH <- matrix(NA_real_, nrow = length(K_grid), ncol = n_exp)
+   for (j in 1:n_exp) {
+     pars <- results$details[[j]]
+     F0   <- as.numeric(CurrentForwardValues[j])
+     Hj   <- H_vec[j]
+
+     vol_mat_KH[, j] <- sabr_vol(F0, K_grid, Hj, pars$Alpha, pars$Beta, pars$Rho, pars$Nu)
+   }
+
+   # persp needs z as [H x K] because x=H_vec and y=K_grid
+   z_HK <- t(vol_mat_KH)
+
+   persp(x = H_vec, y = K_grid, z = z_HK,
+         xlab = "Maturity (years)", ylab = "Strike", zlab = "Black vol",
+         main = "SABR Implied Volatility Surface",
+         theta = theta, phi = phi, expand = expand, ticktype = ticktype)
+ }
```

6.4 Demonstration

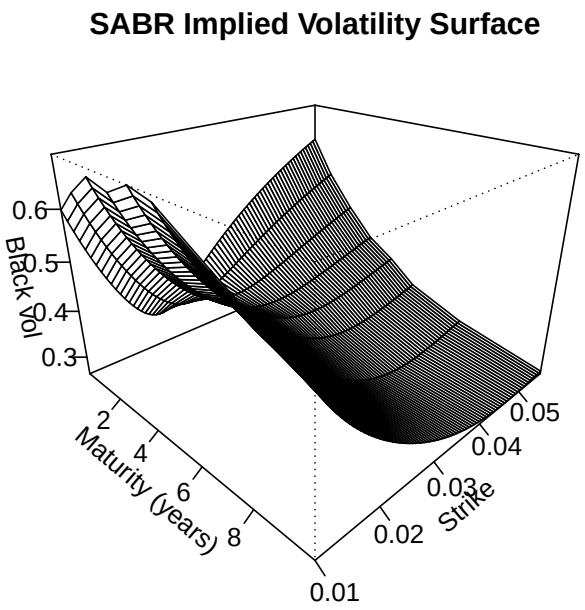
In this last part of the SABR calibration section we will show how the algorithm works. For the sake of this demonstration, we will use some hypothetical data for market implied Black volatilities for European swaptions over a range of strikes. The swaptions have 10-year swaps as the underlying instrument. The rates are expressed in decimals.

```
> SettleDate        <- as.Date('16-Jan-2026', format = "%d-%b-%Y")
> ExerciseDatesChar <- c('16-Mar-2026','16-Jan-2027','16-Jan-2028','16-Jan-2029',
+                        '16-Jan-2030','16-Jan-2031','16-Jan-2033','16-Jan-2036')
> ExerciseDates     <- as.Date(ExerciseDatesChar, format = "%d-%b-%Y")
> YearsToExercise   <- yearfrac(SettleDate, ExerciseDates)
> Maturities        <- c("3M", "1Y", "2Y", "3Y", "4Y", "5Y", "7Y", "10Y")
> Tenor             <- "10Y"
> MarketVolatilities <- matrix(c(
+   # 3M,   1Y,   2Y,   3Y,   4Y,   5Y,   7Y,   10Y
+   57.6, 53.7, 49.4, 45.6, 44.1, 41.1, 35.2, 32.0,
+   46.6, 46.9, 44.8, 41.6, 39.8, 37.4, 33.4, 31.0,
+   35.9, 39.3, 39.6, 37.9, 37.2, 34.7, 30.5, 28.9,
+   34.1, 36.5, 37.8, 36.6, 35.0, 31.9, 28.1, 26.6, # ATM
+   41.0, 41.3, 39.5, 37.8, 36.0, 32.6, 29.0, 26.0,
+   45.8, 43.4, 41.9, 39.2, 36.9, 33.2, 29.6, 26.3,
+   50.3, 46.9, 44.0, 40.0, 37.5, 33.8, 30.2, 27.3
+ ), nrow = 7, byrow = TRUE) / 100
> MarketStrikes     <- matrix(c(
+   # 3M,   1Y,   2Y,   3Y,   4Y,   5Y,   7Y,   10Y
+   1.00, 1.25, 1.68, 2.00, 2.26, 2.41, 2.58, 2.62,
+   1.50, 1.75, 2.18, 2.50, 2.76, 2.91, 3.08, 3.12,
+   2.00, 2.25, 2.68, 3.00, 3.26, 3.41, 3.58, 3.62,
+   2.50, 2.75, 3.18, 3.50, 3.76, 3.91, 4.08, 4.12, # ATM
+   3.00, 3.25, 3.68, 4.00, 4.26, 4.41, 4.58, 4.62,
+   3.50, 3.75, 4.18, 4.50, 4.76, 4.91, 5.08, 5.12,
+   4.00, 4.25, 4.68, 5.00, 5.26, 5.41, 5.58, 5.62
```

```
+ ), nrow = 7, byrow = TRUE) / 100
> # Current Forward Rates and ATM Vols across the maturities
> CurrentForwardValues <- MarketStrikes[4,]
> ATMVolatilities      <- MarketVolatilities[4,]
> results_calibration <- calibrate_sabr_surface(
+                                    SettleDate = SettleDate,
+                                    ExerciseDates = ExerciseDates,
+                                    MarketStrikes = MarketStrikes,
+                                    MarketVolatilities   = MarketVolatilities,
+                                    CurrentForwardValues = CurrentForwardValues,
+                                    beta  = 0.5,
+                                    start = c(0, 0.5),
+                                    include_atm_in_fit = FALSE)
> plot_smile(j = 3,
+            results_calibration,
+            SettleDate, ExerciseDates,
+            MarketStrikes,
+            MarketVolatilities,
+            CurrentForwardValues)
```




```
> plot_surface(results_calibration,
+              SettleDate,
+              ExerciseDates,
+              MarketStrikes,
+              CurrentForwardValues,
+              n_strike = 80)
```



7 Conclusion

In this project, we implemented and analyzed the SABR stochastic volatility model, which generalizes the Black-Scholes framework by allowing volatility to be stochastic. Our implementation includes:

- Simulation of correlated Brownian motions using Cholesky decomposition,
- Euler-Maruyama discretization of the SABR SDEs with special handling for different values of β,
- Exact log-updates for the $\beta = 1$ case and standard Euler approximation for $\beta \neq 1$,
- Verification that SABR reduces to Black-Scholes when $\beta = 1$ and $\nu = 0$.
- Practical calibration approach for the SABR model, using the Hagan et al. approximate closed-form formula to compute Black implied volatilities and fit the volatility smile and surface across strikes and maturities

Our numerical experiments demonstrated:

- The SABR model produces realistic stochastic volatility paths that remain strictly positive,
- The correlation parameter ρ successfully captures the leverage effect observed in equity markets,
- The elasticity parameter β controls how volatility scales with the asset price level,
- Our implementation correctly replicates Black-Scholes behavior in the appropriate limit.

The SABR model is widely used in practice due to its flexibility in matching observed volatility smiles and the existence of closed-form approximations for option prices [Hagan et al., 2002]. The model's ability to capture the relationship between price levels and volatility through the β parameter, combined with the correlation structure between price and volatility dynamics, makes it particularly suitable for interest rate and foreign exchange derivatives markets.

References

P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2004.

P. S. Hagan, D. Kumar, A. S. Lesniewski, and D. E. Woodward. Managing smile risk. *Wilmott Magazine*, pages 84–108, September 2002.

S. E. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models*. Springer, New York, 2004.