Quant

Rodolfo Pietrasanta

September 2025

1 Introduction

This note contains some small quantitative finance projects, aiming to cover all the theoretical and implementation details needed to achieve the goal. In particular, an implied volatility surface interpolation (SVI/SSVI) project is presented. The assumption is that the reader is familiar with calculus, algebra, and basic statistics, but not with financial applications. The note will try to be as clear and complete as possible, avoiding omitting steps in the derivations or in the implementations. If a method is mentioned, it is also explained in detail.

2 Foundations

This project aims to illustrate in details how to perform an Implied Volatility Surface Interpolation. To this end, we start from the basics. At first, we introduce the so called Wiener process.

2.1 Wiener process

The aim of the Wiener process (or standard Brownian Motion) is to model the behavior of a variable whose nature is random. In other words, the evolution of a variable in time if its variation is random. Let's indicate the variable $W(t)$ as W_t for shortness. W_t is a Wiener process if it is characterized by a set of properties:

- It holds that $W_{t+s} - W_t = W(t+s) - W(t) \sim N(0, s)$. This is the most characteristic property. The difference in value of W between two time instants is random and distributed as a Normal variable with 0 mean and a variance equal to the size of the time interval between the two instants.
- The random increments are all independent. Given any two increments $\Delta W_1 = W(t_1) - W(t_1^*) \sim N(0, t_1 - t_1^*)$ and $\Delta W_2 = W(t_2) - W(t_2^*) \sim N(0, t_2 - t_2^*)$, then ΔW_1 and ΔW_2 are independent.
- The trajectories in time are almost surely continuous (the term "almost surely" means that it happens with probability 1). This can be proved to be true with the Kolmogorov continuity theorem, which actually follows from the other assumptions.
- $W(0) = W_0 = 0$ almost surely. This latter hypothesis is more of a practical convention than a fundamental property, but it is still useful in most applications.

Remark 2.1 *Under these assumptions, it follows that:*

$$dW_t = W_{t+dt} - W_t \sim N(0, dt) = \sqrt{dt} N(0, 1). \tag{1}$$

This means that the infinitesimal increment of a Wiener process is random with variance dt, where dt is the infinitesimal increment in time.

Remark 2.2 *From 2.1 we can try to compute the expected value of dW_t and dW_t^2 that will be useful in the following sections.*

$$\mathbb{E}[dW_t] = 0 \qquad \mathbb{E}[dW_t^2] = dt. \tag{2}$$

The first comes from the fact that $dW_t \sim N(0, dt)$, so the expected value is just the mean, which is 0. The second comes from the definition of the expected value of $f(x)$ with $x \sim X$ with probability density $\phi(x)$:

$$\mathbb{E}[f(x)] = \int_S f(x)\phi(x)dx \tag{3}$$

S is the support of X, if $X \sim N(\mu, \sigma^2)$ then $S = \mathbb{R}$. It follows that, for this case, $f(x) = x^2$ and $x \sim \sqrt{t}N(0,1)$:

$$\mathbb{E}[dW_t^2] = \int_{\mathbb{R}} x^2 \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = 1 \tag{4}$$

The last equality is provable via integration by parts.

2.2 Itô's lemma

In this section, we present and prove Itô's lemma, which can be regarded as the stochastic analogue of the Taylor expansion.

Theorem 1 (Itô's lemma) *Consider a Wiener process (or standard Brownian motion) dW_t and a function $f(t,x) : \mathbb{R} \times [0,\infty] \to \mathbb{R}$ at least once differentiable in t and twice in x:*

$$dS_t = \mu_t dt + \sigma_t dW_t. \tag{5}$$

Then

$$df(t,S_t) = f_t(t,S_t)dt + f_x(t,S_t)dS_t + \frac{1}{2}f_{xx}(t,S_t)\sigma_t^2 dt \tag{6}$$

with:

- $f_t = \frac{\partial}{\partial t} f$
- $f_x = \frac{\partial}{\partial x} f$
- $f_{xx} = \frac{\partial^2}{\partial x^2} f$.

Proof: Using the Taylor expansion of f up to order 2 we get:

$$\begin{aligned} f(t+dt, S_{t+dt}) = f(t,S_t) + \\ f_t(t,S_t)dt + f_x(t,S_t)dS_t + \frac{1}{2}f_{tt}(t,S_t)dt^2 + \frac{1}{2}f_{xx}(t,S_t)dS_t^2 + f_t(t,S_t)f_x(t,S_t)dtdS_t \end{aligned} \tag{7}$$

It holds that: $dS_t^2 = \mu_t^2 dt^2 + \sigma_t^2 dW_t^2 + 2\mu_t\sigma_t dW_t dt$. The term $dt^2 \approx 0$.The term $dW_t dt \sim dt^{\frac{3}{2}} N(0,1) \approx 0$. The only surviving second-order term is dW_t^2, which can be proved (under the standard convention) to be $dW_t^2 = dt$. Moving $f_t(t,S_t)$ to the left-hand side of the equation and collecting the remaining terms, we get:

$$df(t,S_t) = f(t+dt, S_{t+dt}) - f(t,S_t) = f_t(t,S_t)dt + f_x(t,S_t)(\mu_t dt + \sigma_t dW_t) + \frac{1}{2}f_{xx}(t,S_t)\sigma_t^2 dt. \tag{8}$$

■

The equation can be further simplified by replacing dS_t:

$$df(t,S_t) = [f_t(t,S_t) + \mu_t f_x(t,S_t) + \frac{1}{2}\sigma_t^2 f_{xx}(t,S_t)]dt + \sigma_t f_x(t,S_t)dW_t \tag{9}$$

2.2.1 Examples

In this section, some examples of the usage of Itô's lemma are provided.

3 Black-Scholes model

In this section, we derive and comment the Black-Scholes (BS) model, necessary to continue with the rest of the notions.

The BS model can be derived starting from Itô's lemma. In particular, we assume an underlying with the following dynamics:

$$dS_t = \mu S_t dt + \sigma S_t dW_t. \tag{10}$$

This means that the drift $\mu_t = \mu S_t$ and the variance $\sigma_t = \sigma S_t$ are both functions of the underlying S_t.

Remark 3.1 *μ and σ are supposed **constants**.*

Let's consider now a portfolio Π that contains both an option on the underlying $V(t, S_t)$ and a certain quantity δ of the underlying itself:

$$\Pi = V(t, S_t) - \delta S_t \tag{11}$$

Remark 3.2 *Note that, for simplicity, the underlying is assumed to be infinitely divisible. In practice, it is impossible to buy an arbitrarily small quantity of underlying.*

We can now take the differential of the portfolio:

$$d\Pi = dV + \delta dS_t. \tag{12}$$

As $V(t, S_t)$ is a generic function, we can apply the Itô's lemma:

$$\begin{aligned} d\Pi &= V_t dt + V_s dS_t + \frac{1}{2}\sigma_t^2 V_{ss} dt - \delta dS_t = \\ &= V_t dt + V_s dS_t + \frac{1}{2}\sigma^2 S_t^2 V_{ss} dt - \delta dS_t. \end{aligned} \tag{13}$$

We recall that, in this context, we will write:

- $V_t = \frac{\partial}{\partial t} V(t, S_t)$
- $V_s = \frac{\partial}{\partial S_t} V(t, S_t)$
- $V_{ss} = \frac{\partial}{\partial S_t^2} V(t, S_t)$

If we chose $\delta = \frac{\partial V}{\partial S_t}$, we can cancel a term. This is called ***delta hedging***. The name comes from the fact that the quantity $\frac{\partial V}{\partial S_t}$ is called delta and that it can be explicitly computed as we will see later. This creates a sort of automatic feedback that tells us how much of the underlying it is convenient to own to compensate for the effect of the variations of the underlying itself on the option.

Once the term $V_s dS_t$ is canceled, we can compare the portfolio with the risk-free exponential growth:

$$d\Pi = r\Pi dt = (V_t + \frac{1}{2}\sigma^2 S_t^2 V_{ss})dt. \tag{14}$$

Remark 3.3 *The first equality is an ODE whose solution is $\Pi(t) = \Pi(0)e^{rt}$.*

We can now simplify dt on both sides and replace $\Pi = V - V_s S_t$, obtaining:

$$V_t + \frac{1}{2}\sigma^2 S_t^2 V_{ss} - rV + rS_t V_s = 0. \tag{15}$$

This latter equation is called the Black-Scholes equation. It is a partial differential equation (PDE). It can be proven that, after some change of variables and some algebraic manipulations, the equation can be reduced to a heat/diffusion PDE. **Proof**

We are now interested in the solutions of 15. Luckily, a closed form solution exists.

Theorem 2 *Consider the partial differential equation (15) with the terminal condition:*

$$V(T, S) = max(S - K, 0) = (S - K)^+, \tag{16}$$

where K is the strike value of the option. Consider that the underlying at $t = 0$ is S_0. Then, the function:

$$C = S\phi(d_1) - Ke^{-rT}\phi(d_2), \tag{17}$$

with the parameters:

$$d_1 = \frac{ln(\frac{S_0}{K}) + (r + \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}}, \quad d_2 = d_1 - \sigma\sqrt{\tau}, \quad \tau = T - t, \tag{18}$$

and the function:

$$\phi(x) = \mathbb{P}(Z < x) \text{ with } Z \sim N(0, 1), \tag{19}$$

is a solution to the PDE.

Proof:

Remark 3.4 *The solution function is called C as the terminal condition $V(T, S) = (S - K)^+$ corresponds to a call option.*

Remark 3.5 *In the solution, S is left without any subscript as it is treated as an independent variable. The right way to see the solution is "this is the cost at time t and if the price is S. The result is a surface for all values of S.*

3.1 The Greeks

4 The Implied Volatility

The main limitation of the Black-Scholes model is that the variance is constant and supposed to be known. In real scenarios, this is not true, making the model hard to be directly applied to option pricing.

However, the model is still used as a common baseline for the computation of the ***implied volatility*** (IV). Supposing to know the price C_{mkt} of an option for a certain strike and maturity, the corresponding IV is the value of σ that, if put into the Black-Scholes machinery, gives back the price C_{mkt}. In simpler words: "*what volatility do I have to plug into the Black-Scholes equation to get the price that I read on the market?*".

$$\sigma_{IV} = \{\sigma : C_{BS}(S_0, r, K, T, \sigma) = C_{mkt}\}. \tag{20}$$

For European vanilla option, we can safely say that the IV is unique for every combination of the other parameters.

To practically compute the implied volatility, there is no direct, inverse formula of (17). Luckily, the function is continuous, so we can apply a classic root-finding algorithm such as the bisection method to approximate it numerically.

4.1 IV computation example

Let's consider a call option with a price of $S = 100$, strike $K = 118$, risk-free rate $r = 0.019$, and maturity $T = 2$ years. We read a market price $C_{mkt} = 20.19$.

Let's start with the basic utility functions:

```
import numpy as np
import math

#Normal cumulative density function
def N(x):
    return 0.5*(1+math.erf(x/np.sqrt(2)))

#Call option BS price
def BS_call(S,K,T,sigma,r):
```

```
        #If the time is now or in the past, it returns the cash in amount
        if T <= 0:
            return max(S-K, 0.0)
        #If the volatility is zero, the evolution of the underlying is
            deterministic
        if sigma <= 0:
            return max(S - K*np.exp(-r*T), 0.0)
        d1 = (np.log(S/K) + (r + 0.5*sigma*sigma)*T) / (sigma*np.sqrt(T))
        d2 = d1 - sigma*np.sqrt(T)
        return S*N(d1) - K*np.exp(-r*T)*N(d2)
```

The function $N(x)$ defines the cumulative density function of the normal distribution. The choice of math.erf is done to avoid scypy or other libraries that could be slower or cumbersome to import.

The function BS_call takes as inputs the price S, strike K, maturity T, volatility σ and risk-free rate r. For this example, the dividends q are considered to be zero. The function manages the limit cases where the maturity is the present moment or in the past and sigma to be zero (or negative in case of input mistake).

We can now define the function that approximates the Implied Volatility using bisection. For further details on the method, see the appendix.

```
def ImpliedVol(S,K,T,r,C):

    #Check no arbitrage in the price for the given parameters
    lb = max(S - K*np.exp(-r*T), 0.0)
    ub = S
    if C < lb - 1e-12 or C > ub + 1e-12:
        return np.nan

    #Compute the initial bracket
    a, b = 1e-8, 5.0  # [~0%, 500%]
    fa = BS_call(S,K,T,a,r) - C
    fb = BS_call(S,K,T,b,r) - C

    #Make the initial bracket larger if necessary
    it = 0
    while fa*fb > 0 and b < 10.0 and it < 30:
        b *= 2.0
        fb = BS_call(S,K,T,b,r) - C
        it += 1
    if fa*fb > 0:
        return np.nan  # Initial bracketing was not possible

    #Halving process taking the midpoint and excluding the point with the same
        sign
    #Halving 50 times times makes the bracket 1.126e15 times smaller.
    #An early exit condition is set at 1e-8.
    for _ in range(50):
        m = 0.5*(a+b)
        fm = BS_call(S,K,T,m,r) - C
        if fa*fm <= 0:
            b, fb = m, fm
        else:
            a, fa = m, fm
        if b - a < 1e-8:
            break
    return 0.5*(a+b)
```

The function ImpliedVol takes as inputs the price S, the strike K, the maturity T, the risk-free rate r, and the market price $C = C_{mkt}$.

The first part of the code manages prices out of the no-arbitrage bounds, initializes the bracket, and returns NaN if not possible.

We can now initialize the parameters and run the algorithm.

```
S = 100
K = 118
T = 2
r = 0.019
Cmkt = 20.19

IV = ImpliedVol(S,K,T,r,Cmkt)
```

We obtain an implied volatility of $\sigma_{IV} \approx 0.4466$.

5 SVI interpolation

The aim now is to have a smooth analytic curve for the implied volatility values as a function of the strikes. To this aim, we have to define some more quantities. The first is the *log moneyness*, which will replace the strikes in the interpolation.

$$x = ln(\frac{K}{F}) \quad \text{where} \quad F = Se^{rT}. \tag{21}$$

The strikes are normalized over the future of the underlying. The log is taken to both make the behavior of the curve more evident and to set 0 as the value of $K = F$.

Since T appears in practical implementations, the log moneyness becomes a matrix, as opposed to the strikes that were listed in a vector. Each row corresponds to a value of T.

The second quantity is the *total variance*.

$$w(K,T) = \sigma_{IV}^2 T. \tag{22}$$

This is done to avoid the divergence (as $1/\sqrt{T}$) of σ_{IV} for $T \to 0$. As for the log moneyness, also the values of total variance are in a matrix with strikes and maturities.

The aim of SVI is to interpolate w as a function of x for a given T. Gatheral proposed the following model:

$$w_{SVI}(x) = a + b\left[\rho(x-m) + \sqrt{(x-m)^2+\sigma}\right]. \tag{23}$$

In this model, a, b, ρ, m, σ are parameters to be identified. The following constraints hold on the parameters:

- $a \geq 0$
- $b \geq 0$
- $\sigma > 0$
- $\rho \in (-1, 1)$

Remark 5.1 *Note that σ in this context is different from the volatility of the underlying, it is just a parameter that happened to have the same name.*

To obtain the function, we need, for a given maturity, a set of strikes and the corresponding market prices. We can then run an optimization algorithm on the parameters to find the optimal parameters.

In this document, a Gauss-Newton-like algorithm will be applied. We start from the loss function:

$$L = \frac{1}{2}\sum_i [w_i - w_{SVI}(x_i)]^2 \tag{24}$$

where $w_i = w(K_i, T^*)$ or the total variance at the strike K_i and fixed maturity T^*. The same holds for the log moneyness x_i. We can now compute the partial derivatives of the model with respect to its parameters:

- $\frac{\partial}{\partial a} w_{SVI} = 1$
- $\frac{\partial}{\partial b} w_{SVI} = \rho(x-m) + \sqrt{(x-m)^2+\sigma}$
- $\frac{\partial}{\partial \rho} w_{SVI} = b(x-m)$
- $\frac{\partial}{\partial m} w_{SVI} = -b\left(\rho + \frac{x-m}{\sqrt{(x-m)^2+\sigma}}\right)$
- $\frac{\partial}{\partial \sigma} w_{SVI} = \frac{b}{2\sqrt{(x-m)^2+\sigma}}$

And build the Jacobian $J_i = \begin{bmatrix} \frac{\partial}{\partial a} & \frac{\partial}{\partial b} & \frac{\partial}{\partial \rho} & \frac{\partial}{\partial m} & \frac{\partial}{\partial \sigma} \end{bmatrix}^T w_{SVI}(x_i)$.

We generate one Jacobian J_i for each strike and stack them in columns, obtaining a Jacobian matrix J. Defining the vector:

$$e = w - w_{SVI} \tag{25}$$

where each entry is $w_i - w_{SVI}(x_i)$, and the parameters vector θ:

$$\theta = \begin{bmatrix} a & b & \rho & m & \sigma \end{bmatrix}^T \tag{26}$$

we can proceed to the application of the Gauss-Newton algorithm:

$$\theta_{k+1} = \theta_k - (J^T J + \lambda I)^{-1} J^T e. \tag{27}$$

Remark 5.2 *The SVI total variance is affine in a, i.e. $\partial w/\partial a = 1$ and $\partial^2 w/\partial a^2 = 0$. Consequently, the exact Hessian has a zero first row/column in the a direction, and $J^\top J$ carries no curvature information along a. It is therefore convenient to eliminate a analytically and optimize only over $\phi = (b, \rho, m, \sigma)$. Writing*

$$w_{SVI}(x;\theta) \;=\; a \;+\; f(x;\phi), \qquad f(x;\phi) \;=\; b\Big[\rho\,(x-m) + \sqrt{(x-m)^2+\sigma^2}\Big],$$

the optimal offset at iteration k is obtained in closed form as

$$a^{k+1} \;=\; \overline{w} \;-\; \overline{f(k;\phi^k)},$$

where, for any vector $x = (x_i)_{i=1}^n$, $\overline{x} := \frac{1}{n}\sum_{i=1}^n x_i$. This reduces the normal equations from 5×5 to 4×4 and typically improves conditioning, robustness, and speed.

The logic behind this algorithm is to approximate the Hamiltonian $H \approx J^T J$. The parameter λ can be fixed or dynamic. For small λ, the behavior is closer to a pure Newton method. For higher values of λ, the method is closer to a pure gradient descent. This algorithm provides faster convergence compared to classical gradient descent.

Remark 5.3 *Note that λI also avoids the inverted matrix from being singular, which is the case as the second derivatives with respect to a are all 0, making the true Hamiltonian singular and the approximation $J^T J$ close to singularity.*

Remark 5.4 *The analytic Hamiltonian could be computed for even faster convergence by taking the second derivatives of the SVI model with respect to the parameters.*

The process is then repeated for all the maturities T. We obtain a set of optimal parameters for each different

5.1 SVI computation example

In this example, we will consider:

$$T = \begin{bmatrix} 0.25 & 0.5 & 1.0 & 2.0 \end{bmatrix} \in \mathbb{R}^4$$

and $K \in \mathbb{R}^{20}$ equally spaced values from 80 to 118 with spacing 2. The matrix of market prices $C_{mkt}\mathbb{R}^{4,20}$ is left in the code for brevity.

We define the following function to check for the monotonicity of the prices and avoid arbitrage.

```
def monotoneCheck(C, tol = 1e-5):
    s = np.shape(C)
    monotone = True
    for i in range(s[0]):
        for j in range(1,s[1]):
            if C[i,j]-C[i,j-1]>tol:
                monotone = False
                break
    return monotone
print('Check monotonicity with the strikes: ' + str(monotoneCheck(C)))
print('Check monotonicity with the maturities: ' + str(monotoneCheck(-C.T)))
```

The following two functions are defined to compute the model SVI and its derivatives:

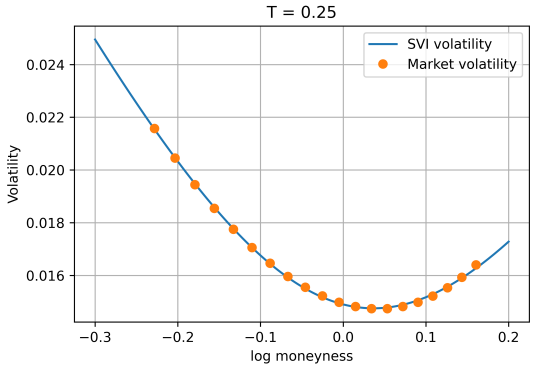
```
def wSVI(k, a, b, rho, m, sigma):
    k = np.asarray(k)
    s = np.sqrt((k - m)**2 + sigma)
    return a + b * (rho * (k - m) + s)

def jacobian_terms(k, b, rho, m, sigma):
    k = np.asarray(k)
    s = np.sqrt((k - m)**2 + sigma)
    dw_da    = np.ones_like(k)
    dw_db    = rho*(k - m) + s
    dw_drho = b*(k - m)
    dw_dm    = b * (-rho + (m - k)/s)
    dw_dsigma = b * (1.0 / (2.0 * s))
    return dw_da, dw_db, dw_drho, dw_dm, dw_dsigma
```

And the following to run the optimization algorithm and obtain the optimal parameters:

```
def SVI_fit(k, w_data, iters=100):

# Init
a, b, rho, m, sigma = 0.01, 0.1, -0.1, 0.0, 0.05  # valori ragionevoli
loss_hist = []

for _ in range(iters):
    w_model = wSVI(k, a, b, rho, m, sigma)
    r = w_model - w_data  # residui
    # MSE
    loss = 0.5 * np.dot(r, r)
    loss_hist.append(loss)
    if np.log(loss_hist[-1]/loss_hist[0]) < -9:
        print('The optimization converged.')
        break

    # J^T r
    dw_da, dw_db, dw_drho, dw_dm, dw_dsigma = jacobian_terms(k, b, rho, m,
        sigma)
    J = np.column_stack([dw_db, dw_drho, dw_dm, dw_dsigma]) #n x 5 matrix
    lam = 0.04
    param = np.array([b,rho,m,sigma])
    param = param - np.linalg.pinv(J.T@J+lam * np.eye(J.shape[1]))@(J.T)@r

    b = param[0]
    rho = param[1]
    m = param[2]
    sigma = param[3]
    a = np.mean(w_data)-np.mean(wSVI(k, a, b, rho, m, sigma))

    # clip
    a = max(a,0)
    b    = max(b, 1e-6)
    sigma = max(sigma, 1e-8)
    rho = np.clip(rho, -0.999, 0.999)

    cap = 2.0/(1.0+abs(rho))
    if b > cap: b = cap

return a, b, rho, m, sigma, loss_hist
```

Remark 5.5 *Note that the constraints on the parameters can be integrated into the optimization by a change of variables. For example, $\rho = tanh(\rho^*)$ forces it to be in range. The exponential function can be used for parameters that have to be positive.*

The following figures show the obtained SVI functions for all the different maturities.



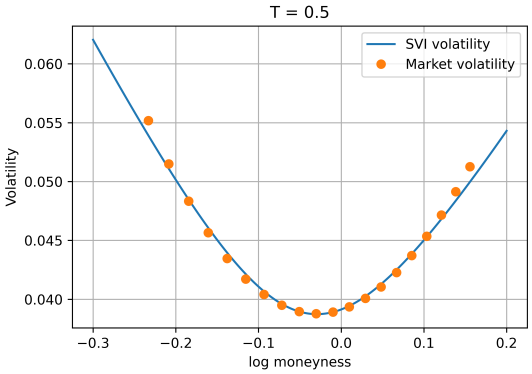
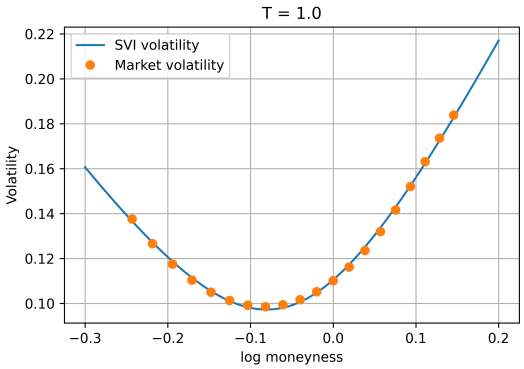


Figure 1



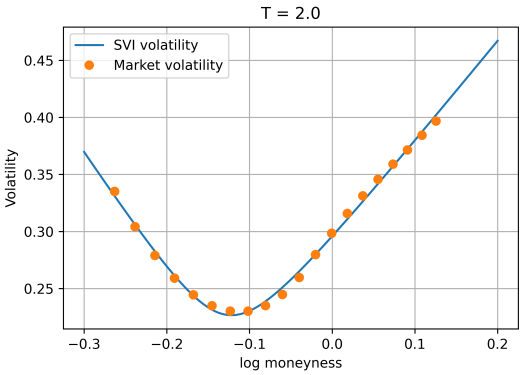


Figure 2