Interest Rate Volatility Cube: Construction And Use.

Pat Hagan, Michael Konikov

20th July 2004

1 Introduction.

This document describes how to create a volatility cube object. The volatility cube object is an object that takes as input a yield curve, cap volatility matrix, swaption volatility matrix, and, possibly, eurodollar future option (EDFO) prices, and is able to compute a swaption volatility for any given triplet of option tenor, swap tenor, and strike. This object will be used in all the nonstandard vanilla deal pricing and all the exotic deal pricing.

Figure 1 explains the inputs and the geometry of the volatility cube. Caplet volatilities that will be stripped from the given cap volatilities fill one face of the cube, since caplets are a special case of swaptions with swap tenor equal typically three months (at least for USD). Swaption volatility matrix, on the other hand, is given for all the swaption tenors both in option and swap directions, but only at-the-money, i.e, only for a strike equal corresponding swap rate. From this inputs, possibly using EDFO prices instead of short term caplets, we need to fill the whole cube of swaption volatilities for all option terms, swap terms and strikes.

The construction of the volatility cube object consists of the following steps. If we include eurodollar future option prices in our construction (EDFOs for short), we first calculate implied Black volatilities from EDFOs prices. Then we interpolate those along the strike direction on the set of cap strikes, since volatilities obtained from EDFOs will be used for short maturities in the cap stripping procedure. Lastly, for each strike we fit a

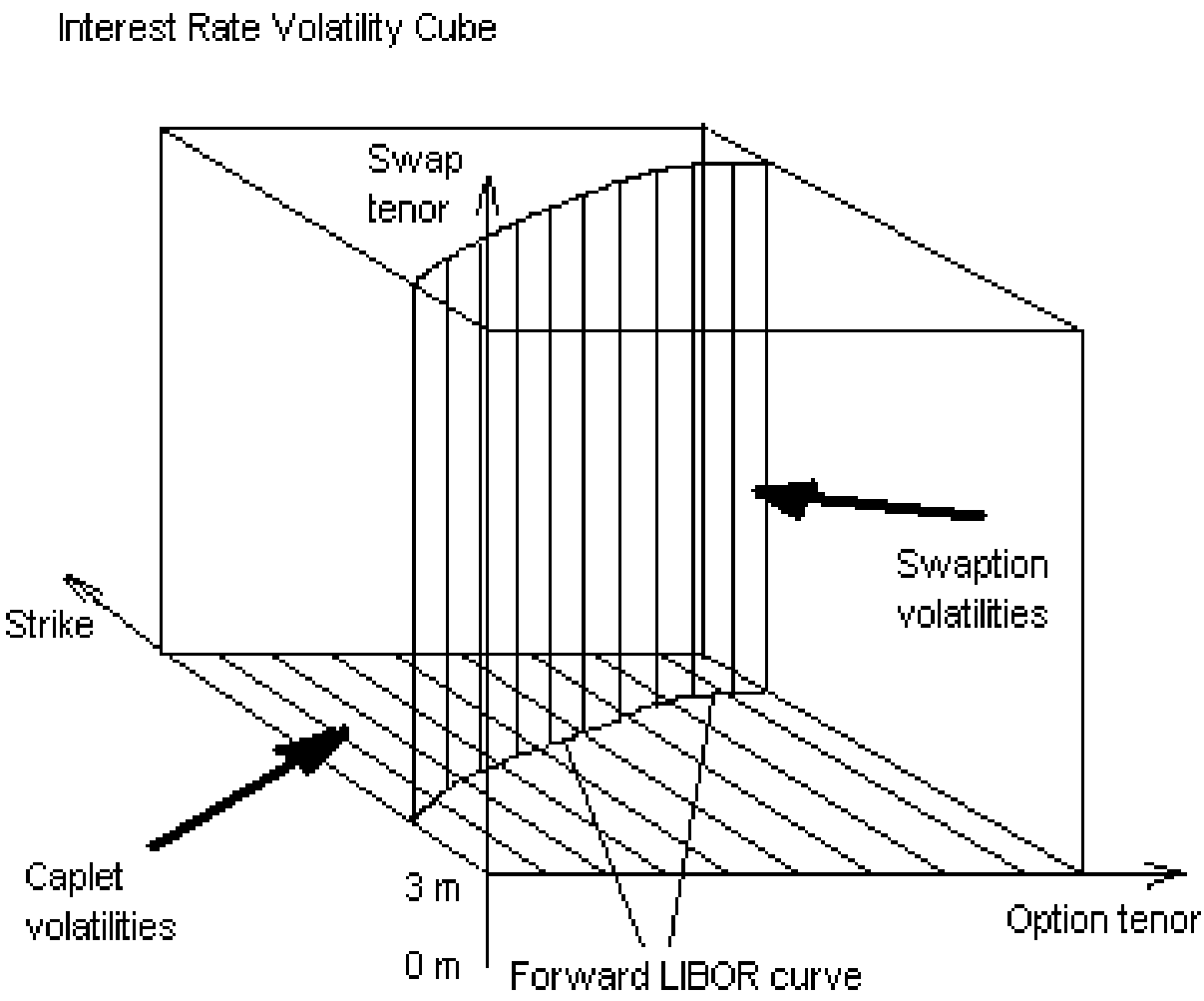

Figure 1: Interest Rate Volatility Cube

corresponding functional form that would be used in the cap stripping procedure. These functional forms will be used as the first function for each corresponding maturity interpolator.

Next, for each cap strike, we strip constant caplet volatilities from given cap volatilities inverting numerically Black's pricing formula. If the specified stripping method is the constant caplet volatility one, we are done with stripping. Otherwise, we further find volatility values for cap terms and use them as initial guess in optimizational stripping with piece-wise linear or quadratic stripping method. At the same time as we do stripping, we build a collection of maturity interpolators that will be used in the next step.

Next, for each swaption option tenor, we find interpolated volatilities for each of the cap strikes using maturity interpolators. Then we fit these vector of volatilities to a given type of smile (strike) interpolator, building, therefore, a collection of strike interpolators for each swaption option tenor. Lastly, we fill out the whole volatility cube by assigning to swaption volatility grid node $(optionTenor, swapTenor)$ a strike interpolator by taking a cap strike interpolator for a corresponding option tenor, and adjusting one of its parameters to match the at-the-money (ATM) swaption volatility at the node.

2 Interpolation Spaces.

There are two types of interpolation spaces in which we work. They are related to maturity and strike interpolation methods. In general, one needs to distinguish between functional form of interpolation, in which just a simple deterministic function is used, such as piece-wise polynomial interpolation, for instance, and stochastic interpolation, in which a stochastic model is used, such as CEV or SABR, for instance, as different rules may apply, as we will see, depending on which type given interpolation belongs to.

The two types of spaces in which we work are the strike type spaces and the volatility type spaces, and each volatility cube construction is specified by the pair of such spaces. The strike type applies only to functional form of smile interpolation, since stripping is done per strike, and stochastic interpolators do not care about strike space as well. If strike type is absolute strike, we use strikes as they are, and if it is relative strike, we use moneyness, i.e., $\ln(K/F)$, where F is the corresponding at-the-money rate, and K is given strike. To illustrate how it works, lets take a linear interpolation as

an example. In absolute space we would have

$$V = \frac{V_i(K_{i+1} - K) + V_{i+1}(K - K_i)}{K_{i+1} - K_i}, \tag{1}$$

where as in relative space we would have

$$V = \frac{V_i \left(\ln(K_{i+1}/F) - \ln(K/F)\right) + V_{i+1} \left(\ln(K/F) - \ln(K_i/F)\right)}{\ln(K_{i+1}/F) - \ln(K_i/F)}, \tag{2}$$

The second type of spaces is the volatility type. We consider three types of volatility: normal, CIR, and Black. They represent volatility for the CEV model:

$$dF_t = \sigma F_t^{\beta} dW_t, \tag{3}$$

corresponding to the 0, $\frac{1}{2}$, and 1 values of β respectively. In practice, we do not convert volatility between these types exactly, since we only interested in nature of volatility here and not the exact model, so, since the original volatilities are always the Black ones ($\beta = 1$), what we do is the following.

$$\begin{aligned} V_{CIR} &\approx V_{Black}(FK)^{\frac{1}{4}} \\ V_{Normal} &\approx V_{Black}\sqrt{FK} \end{aligned} \tag{4}$$

So, we use the approximation on the right hand side of the equations above to go from Black space into normal and CIR ones. Again, this only applies to the functional types of smile interpolation. But here we also need to use the same space in stripping as we use in the smile interpolation, since stripping is a functional type of interpolator. To give an example, if we are in the relative strike space and normal vol space doing a piece-wise linear interpolation, we would use the following formula

$$V = \frac{V_i\sqrt{FK_i} \left(\ln(K_{i+1}/F) - \ln(K/F)\right) + V_{i+1}\sqrt{FK_{i+1}} \left(\ln(K/F) - \ln(K_i/F)\right)}{\sqrt{FK} \left(\ln(K_{i+1}/F) - \ln(K_i/F)\right)}. \tag{5}$$

3 Stripping Caplet Volatility Methods.

In order to build a volatility cube, one of the things we need to do is to extract caplet volatilities from market quoted cap volatilities and to be able

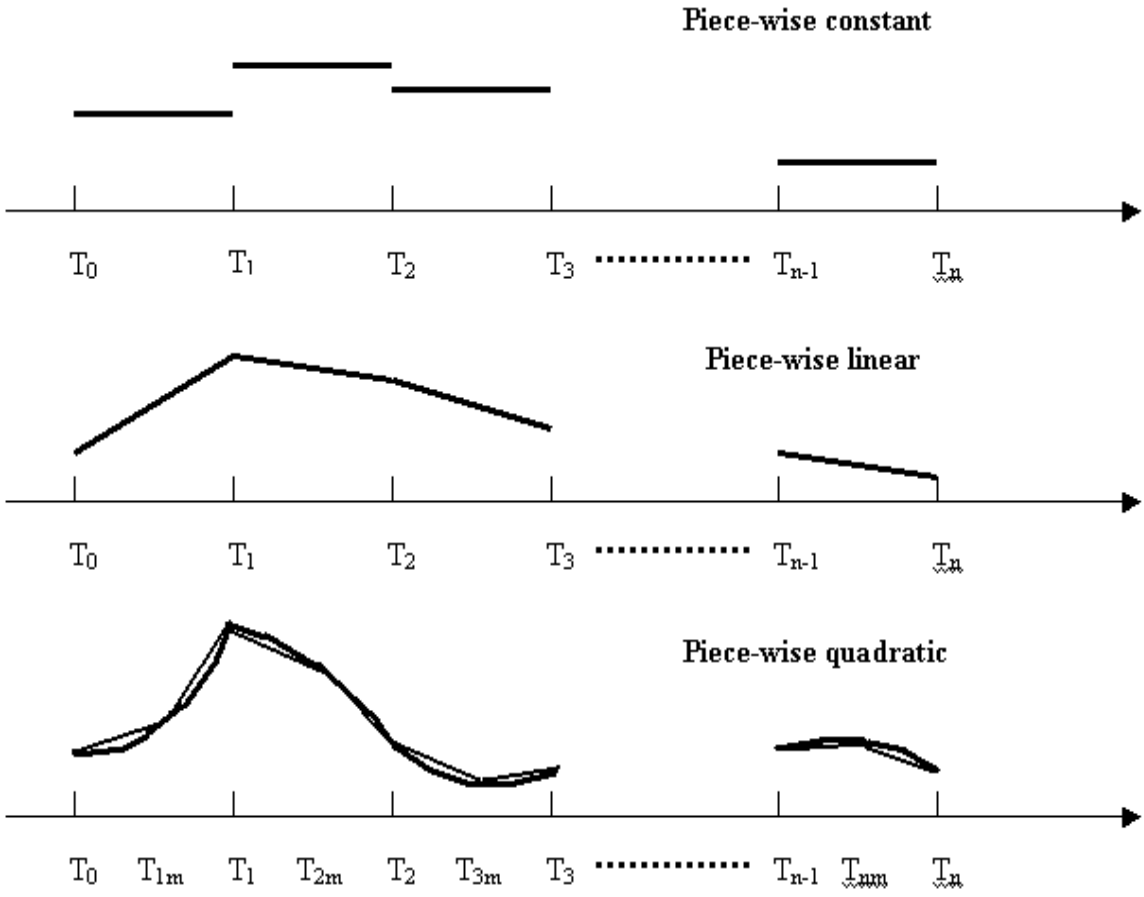

Figure 2: Stripping Methods.

to interpolate in between cap terms in a smooth and consistent with cap volatilities fashion. This process is called stripping cap volatilities. We tested four types of such stripping:

1. Constant caplet volatilities,
2. Piece-wise linear ameliorated caplet volatilities,
3. Piece-wise quadratic inexact ameliorated caplet volatilities,
4. Piece-wise quadratic exact ameliorated caplet volatilities.

The result of stripping is a collection of maturity interpolators (one per strike), where each interpolator consists of term grid and a collection of functions defined on each of the intervals between the neighboring grid points.

3.1 Constant Caplet Volatilities Method.

This is the most basic method to strip cap volatilities. This method will always be used for other methods to get initial guess values for stripping optimization, since all other methods require such an optimization. This is the only method that requires just a series of one-dimensional root solvers. This method is guaranteed to work if the data is consistent, i.e., if all the forward values are above the corresponding intrinsic values. Thus, if this method complains that it cannot properly strip cap volatilities, the data should be questioned. Cap volatility data is quite sensitive, since it arrives in an asynchronous fashion, and far out-of-money or in-the-money data may not be updated often enough. Note that other methods in principal may fail to strip, in which case we recommend to fall back to simpler methods instead. The data may still be questioned for validity. Let $\{T_i\}_{i=0}^n$ be cap terms, where $T_0 = 0$. Let $\{\Sigma_i\}_{i=1}^n$ be corresponding cap volatilities for a given strike K, and $\{\sigma_i\}_{i=1}^n$ be the constant caplet volatilities that we are trying to compute. The stripping is done by the following bootstrapping mechanism. First, we compute cap prices from cap volatilities

$$Cap^K_{[0,T_i]}(K, \Sigma_i) = \sum_{j=1}^{N_i} Caplet^K_{[t_{j-1},t_j]}(\Sigma_i), \tag{6}$$

where $\{t_j\}_{j=1}^N$ are the caplet terms and $\{N_i\}_{i=1}^n$ are such that $t_{N_i} = T_i$. For convenience, let $Cap^K_{[0,T_0]}(K, \Sigma_i) = 0$. Cap terms are usually spaced one year

apart, when as caplet terms are spaced typically three months apart (for USD), although this may depend on currency. To compute caplet prices, we use Black's formula

$$Caplet^K_{[t_{j-1},t_j]}(\sigma) = B(0,t_j)\Delta(t_{j-1},t_j)\left(FN(d_1) - KN(d_2)\right), \tag{7}$$

where $F = F(0,t_{j-1},t_j)$ is forward Libor rate's value, $B(0,t_j)$ is zero bond's value (both can be obtained from today's yield curve), $\Delta(t_{j-1},t_j)$ denotes year fraction between t_{j-1} and t_j, and

$$d_{1,2} = \frac{\ln(F/K)}{\sigma\sqrt{t_{j-1}}} \pm \frac{\sigma\sqrt{t_{j-1}}}{2}. \tag{8}$$

Then we compute forward cap prices:

$$ForwardCap^K_{[T_{i-1},T_i]}(K,\Sigma_i) = Cap^K_{[0,T_i]}(K,\Sigma_i) - Cap^K_{[0,T_{i-1}]}(K,\Sigma_i). \tag{9}$$

It is important to remember that for first cap we don't take into account first caplet interval, since it has volatility zero. Now, to find caplet piece-wise constant volatilities, we need to solve the following equation of one variable

$$ForwardCap^K_{[T_{i-1},T_i]}(K,\Sigma_i) = \sum_{j=N_{i-1}+1}^{N_i} Caplet^K_{[t_{j-1},t_j]}(\sigma_i). \tag{10}$$

Solving these equations for σ_i completes the caplet volatility stripping procedure. In order to get a volatility now for the same strike K and some term T, we find i such that $T_{i-1} < T \leq T_i$, and return σ_i. If there is no such i, i.e., $T > T_n$, we return σ_n.

3.2 Piece-wise Linear Ameliorated Caplet Volatilities Method.

Piece-wise linear ameliorated caplet volatilities method is a method using piece-wise linear continuous functional form. On each interval between two neighboring cap terms, it is a linear function. It is completely determined by its values at cap terms, since for a linear function, knowledge of values at any two points is enough to define the function completely, and, hence, on each interval our function is completely defined by the node values. Thus, we have $N+1$ degrees of freedom ($N+1$ node values) and N cap volatilities to

fit, where N is the number of intervals. This gives us only one free parameter if we want an exact fit, say $f(0)$ - left end value. From our experience, such a fit results in a very unstable functional form, in a sense that it depends largely on the value of the free parameter, and produces rather rough shape for caplet volatilities. What we decided to do instead is to do an inexact fit with an extra penalty term responsible for smoothness, or amelioration, of the resulting caplet surface. The minimization function has the form

$$\begin{aligned} F(f_0, \ldots, f_N) &= \sum_{i=1}^{N} w_i \left(ForwardCap^K_{[T_{i-1},T_i]}(K, \Sigma_i) - ForwardCap^K_{[T_{i-1},T_i]}(K, f) \right)^2 \\ &+ \lambda \sum_{i=2}^{N-1} \left(\beta_{i-1} - \beta_i \right)^2, \end{aligned} \tag{11}$$

where $ForwardCap^K_{[T_{i-1},T_i]}(K, f)$ is the forward caplet price calculated with interpolated volatilities, β_i is the i-th slope, i.e., $\beta_i = \frac{f_i - f_{i-1}}{T_i - T_{i-1}}$, and w_i was taken to be $\frac{1}{T_i}$, although we do realize that it is neither optimal nor unique way to choose weights for this problem. It seems to work fine though in the spot tests that we have performed so far. λ is a parameter controlling how much precision we are willing to trade for additional smoothness of a resulting caplet volatility surface. It is constant during the calibration. In practice, we set it equal to about $10^{-4} - 10^{-3}$, since we do not want to loose too much precision.

3.3 Piece-wise Quadratic Inexact Ameliorated Caplet Volatilities Method.

Piece-wise quadratic inexact ameliorated caplet volatility method is very similar to the previous one. The only difference is that on each interval it is a quadratic function. We parametrize it with values at end points and the mid point of each interval. This is convenient for two reasons. One reason is that we know the range of values for volatilities, hence, it is easier to bound them from below and from above, unlike say the coefficients of quadratic function that can have any values. Another reason is that we need mid values anyway to compute two slopes on each interval: one between left point and mid point and one between mid point and right point. These slopes we will use in the

penalty term in our nonlinear least squares optimization. Since

$$\begin{aligned} f_i &= f_{i-1} + C_1(T_i - T_{i-1}) + C_2(T_i - T_{i-1})^2 \\ f_m &= f_{i-1} + \frac{1}{2}C_1(T_i - T_{i-1}) + \frac{1}{4}C_2(T_i - T_{i-1})^2, \end{aligned} \tag{12}$$

we can derive that

$$\begin{aligned} C_2 &= 2\frac{(f_i - f_{i-1}) - 2(f_m - f_{i-1})}{(T_i - T_{i-1})^2} \\ C_1 &= \frac{f_i - f_{i-1}}{T_i - T_{i-1}} - C_2(T_i - T_{i-1}). \end{aligned} \tag{13}$$

Thus, the parameters of the minimization are $(f_0, f_{1m}, f_1, f_{2m}, \ldots, f_{N-1}, f_{Nm}, f_N)$, and the function we are minimizing is

$$\begin{aligned} F(f) &= \sum_{i=1}^{N} w_i \left(ForwardCap^K_{[T_{i-1},T_i]}(K, \Sigma_i) - ForwardCap^K_{[T_{i-1},T_i]}(K, f) \right)^2 \\ &+ \lambda \sum_{i=1}^{2N-1} \left(\beta_{i-1} - \beta_i\right)^2, \end{aligned} \tag{14}$$

$$\beta_i = \begin{cases} \frac{f_{km} - f_{k-1}}{T_{km} - T_{k-1}}, & i = 2k \\ \frac{f_k - f_{km}}{T_k - T_{km}}, & i = 2k+1, \end{cases}$$

where $T_{km} = \frac{T_{k-1}+T_k}{2}$.

3.4 Piece-wise Quadratic Exact Ameliorated Caplet Volatilities Method.

The only difference between this method and the previous one is that here we are not willing to give up any precision at all. Since we have an extra degree of freedom compare to piece-wise linear interpolator, we use it to fit cap volatilities exactly, using the rest of the parameters for amelioration. The most natural way to achieve this is to use the f_{km}'s for the exact fit and to use f_k's for amelioration. Thus, our parameters are $(f_0, \ldots, f_n)$, and our minimization function is

$$F(f) = \sum_{i=1}^{2N-1} \left(\beta_{i-1} - \beta_i\right)^2, \tag{15}$$

but on each parameter change we fit f_{km}'s to the forward cap prices on each interval. This method has the advantage of being exact and smooth at the same time, but it is computationally very expansive, since besides the global optimization problem, we have to perform one-dimensional root solving every time parameters change, for every interval.

3.5 Choosing Node Values For Non-Piece-Wise Constant Stripping Methods.

All the other stripping method besides the piece-wise constant one are global optimization type of methods. The dimensionality of these optimization problems can be quite high, and it is very important to get a realistic initial guess. For the piece-wise linear stripping, we need to choose the initial node values (values at the cap terms), and for piece-wise quadratic ones, we also need the mid-interval points. The mid points we choose so that the initial functional form of all the quadratics is linear, i.e.,

$$f_{km} = \frac{f_{k-1} + f_k}{2}. \tag{16}$$

To choose the node values, we use method proposed in [4]. We will not discussed it here in detail, and interested reader is refered to [4]. In short, we use the following formulas

$$\begin{cases} f_k = \frac{\sigma_{k-1}(T_k - T_{k-1}) + \sigma_k(T_{k+1} - T_k)}{T_{k+1} - T_{k-1}}, \quad k = 1, \ldots, n-1, \\ f_0 = \frac{3}{2}\sigma_0 - \frac{1}{2}f_1, \\ f_n = \frac{3}{2}\sigma_{n-1} - \frac{1}{2}f_{n-1}. \end{cases}$$

4 Incorporation Of EDFOs.

Currently, we are not using EDFOs in the volatility cube construction, since the data is unavailable, but since this will change in the future, we have a flag that indicates wether we are incorporating EDFOs or not. If EDFO prices are used, we go through the following procedure. First, we convert prices into volatilities inverting Black's formula. Then for each maturity we interpolate these volatilities onto cap strikes using the same smile interpolator as will be used in the rest of the volatility cube construction for consistency. This step is potentially dangerous, since for very short maturities we may

have only volatilities concentrated around the ATM, and when we will try to extrapolate those on the cap strikes, which is a much wider set, we may get some pretty large errors at the wings. One way to deal with this problem is to resort to the constant or linear extrapolation for far out- or in-the-money volatilities.

Next, for each cap strike we fit a functional form to the volatilities using simple linear regression. For consistency functional form should be the same as would be used in the stripping procedure, i.e., constant, linear or quadratic function. We use the obtained functions as the first interval functions for the corresponding maturity interpolators (interval $[0, 1]$). Also, in the stripping procedure, we consider the first interval to be fixed (to have fixed parameters). Thus, we do not perturb these parameters in the optimization, and for constant caplet volatility stripping, we use the values from the first interval instead of the first cap volatility.

5 Smile Interpolation Methods.

We have implemented five smile (strike) interpolation methods in this projects:

1. Piece-wise linear,
2. V-shaped,
3. Hyperbolic,
4. CEV,
5. SABR.

5.1 Piece-wise Linear Method.

The piece-wise linear method is the simplest one, and the only method that fits the original data exactly. The formula to compute interpolated value for a given $x \in [x_i, x_{i+1}]$ given values y_i and y_{i+1} at x_i and x_{i+1} respectively is

$$y = \frac{y_i(x_{i+1} - x) + y_{i+1}(x - x_i)}{x_{i+1} - x_i} \tag{17}$$

where x's and y's depend on the space we are in, x's can be strikes or moneyness, and y's are normal, CIR or Black volatilities respectively. We determine

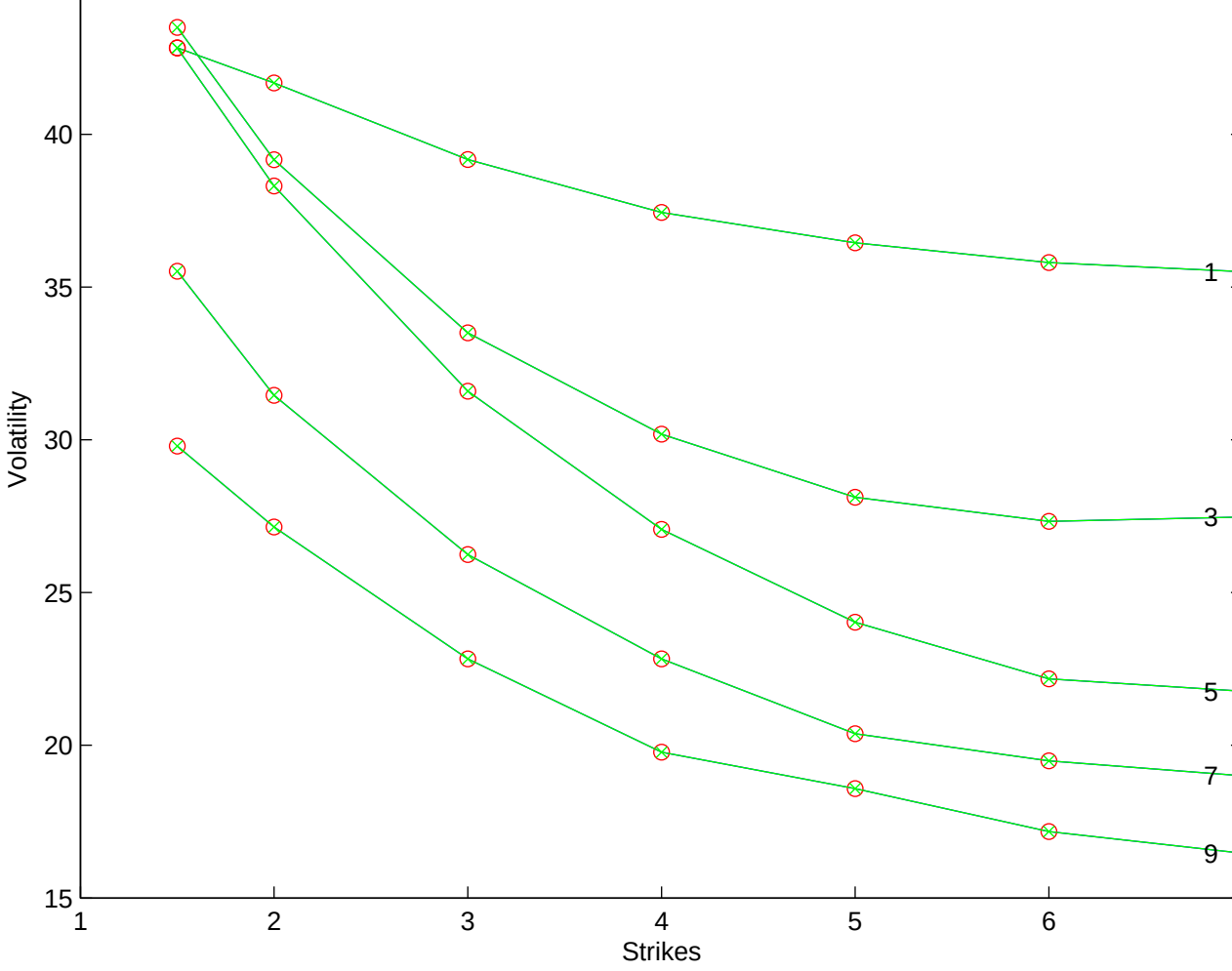

Figure 3: Piece-wise linear method. Blue lines are fitted vols and green lines are the original ones.

the value of i using binary search algorithm described in [5]. To see how extrapolation is handled, please, refer to [4].

This method has no restrictions on the number of strikes needed for its fit.

5.2 V-Shaped Method.

The V-shaped method is not a precise method. We use a nonlinear least squares method described in [5] to find a set of parameters which minimize an L^2-error. As in the previous method, we assume we are given vector of abscesses $\{x_i\}_{i=1}^n$ and vector of values $\{y_i\}_{i=1}^n$. The optimization algorithm used for the nonlinear least squares is a variant of a trust region algorithm designed and implemented by Arun Verma. The function being minimized in this procedure is

$$\sum_{i=1}^{n} w_i \left(y(x_i) - y_i\right)^2 \tag{18}$$

The method has four parameters $(x^*, y^*, \beta_1, \beta_2)$. The formula for y as a function of x is

$$y(x) = \begin{cases} \beta_1(x - x^*) + y^*, & x \leq x^* \\ \beta_2(x - x^*) + y^*, & x \geq x^*. \end{cases}$$

The shape of the function $y(x)$ consists of two rays initiated at the point (x^*, y^*), with left slope β_1 and right slope β_2, hence, the name "V-shaped". Typically, $\beta_1 \leq 0$ and $\beta_2 \geq 0$. The weights used in the calibration are $w_i = \frac{1}{1+(x_i - x^*)^2}$, although, we realize that this is neither optimal in any sense nor unique weighting scheme, and we may try other weighting schemes in the future. It seems to work fine in the few cases that we spot tested.

This method requires at least four strikes per maturity, since we are fitting four parameters. If fewer strikes are given we recommend using other methods instead.

5.3 Hyperbolic Method.

Hyperbolic method is very similar to the V-shaped method described above and, in fact, it is a refinement of the latter. It is not an exact method as well, and is fitted using nonlinear least squares procedure. The relationship

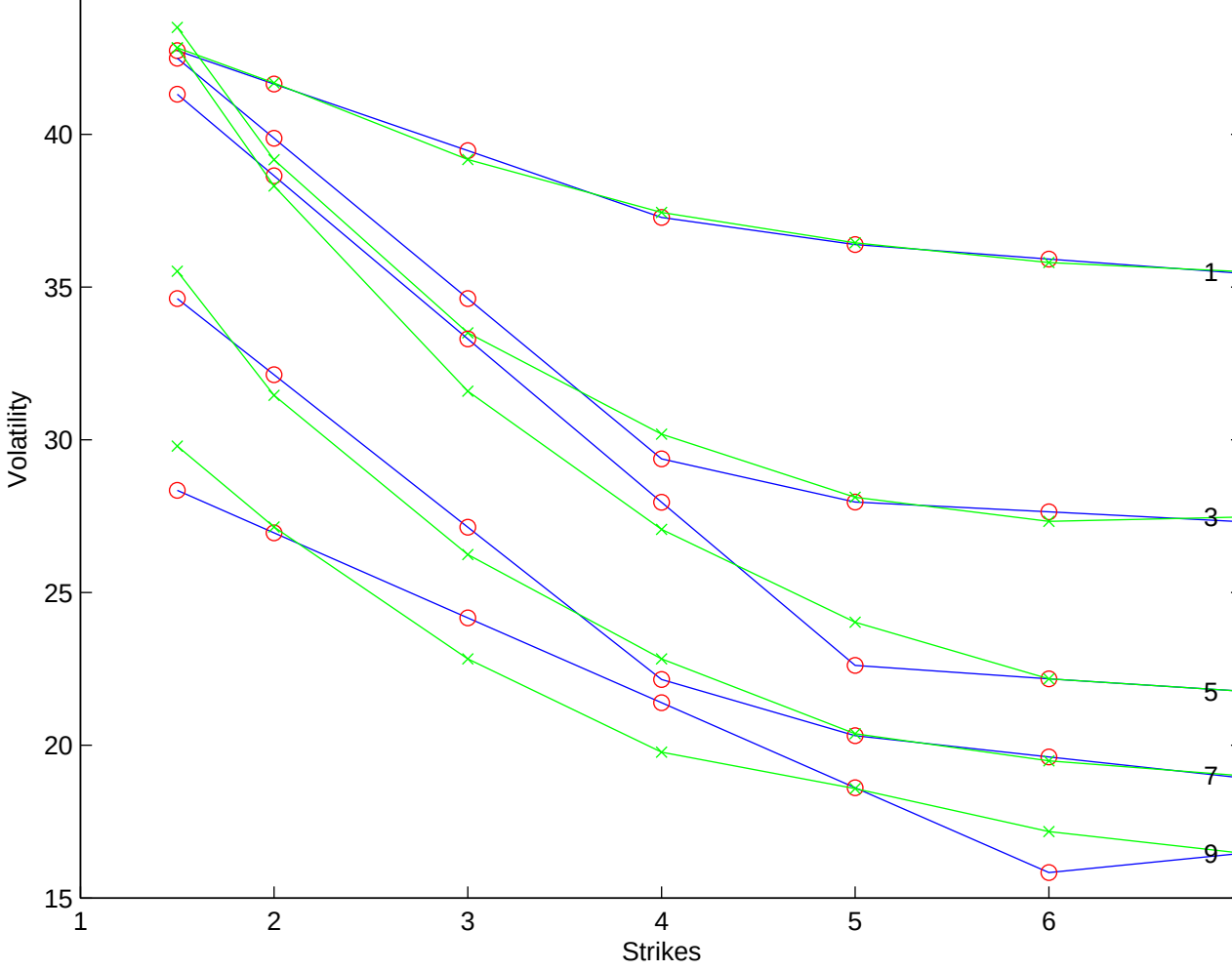

Figure 4: V-shaped method. Blue lines are fitted vols and green lines are the original ones.

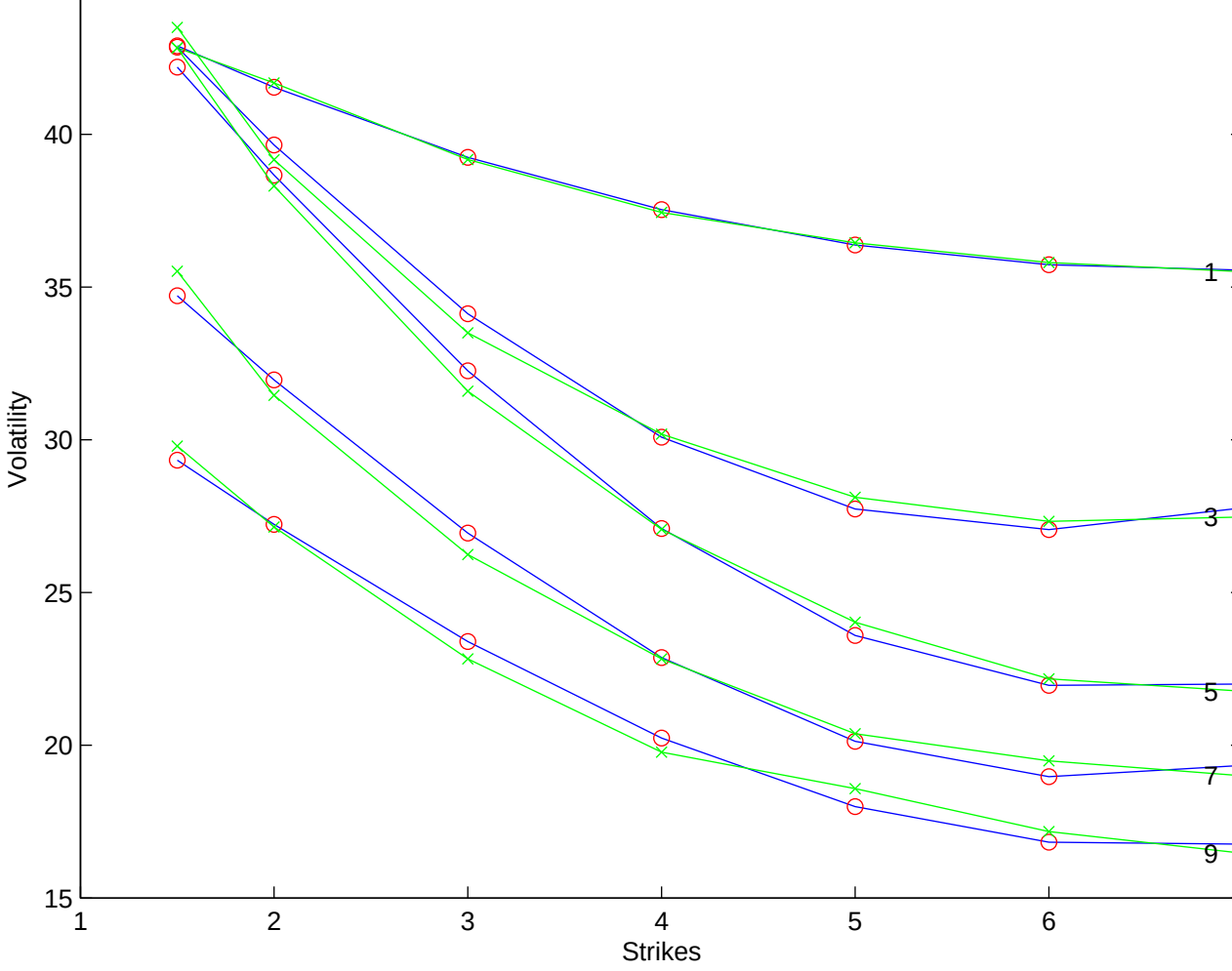

Figure 5: Hyperbolic method. Blue lines are fitted vols and green lines are the original ones.

between x and y is assumed to be

$$y^2 + \beta_1\beta_2(x - x^*)^2 - (\beta_1 + \beta_2)y(x - x^*) = y^{*2}, \tag{19}$$

where $\beta_1\beta_2 < 0$. This is obviously a hyperbola and $y(x^*) = y^*$. If we divide throughout equation by $(x - x^*)^2$, we can rewrite it in the form

$$k^2 - (\beta_1 + \beta_2)k + \beta_1\beta_2 = O\left(\frac{1}{x - x^*}\right), \tag{20}$$

where $k = \frac{y-y^*}{x-x^*}$, which shows that the curve $y(x)$ has two linear asymptotes with exactly the same equations as V-shaped function with the same parameters. Thus, we can think of the hyperbolic method as a smoothed version of the V-shaped method. In order to derive $y(x)$, we just need to solve a quadratic equation, and get:

$$\begin{aligned} y(x) &= \frac{1}{2}\left((\beta_1 + \beta_2)(x - x^*) \pm \sqrt{(\beta_1 + \beta_2)^2(x - x^*)^2 - 4(\beta_1\beta_2(x - x^*)^2 - y^{*2})}\right) \\ &= \frac{1}{2}\left((\beta_1 + \beta_2)(x - x^*) \pm \sqrt{(\beta_1 - \beta_2)^2(x - x^*)^2 + 4y^{*2})}\right) \\ &= \frac{1}{2}\left((\beta_1 + \beta_2)(x - x^*) + \sqrt{(\beta_1 - \beta_2)^2(x - x^*)^2 + 4y^{*2})}\right) \end{aligned} \tag{21}$$

We choose plus sign, since only this sign makes sense. If we set $x = x^*$, we get back y^*, and if $(x - x^*)^2 \gg y^*$, we get

$$y(x) \approx \frac{1}{2}\left((\beta_1 + \beta_2)(x - x^*) + (\beta_2 - \beta_1) \mid x - x^* \mid\right), \tag{22}$$

where the right hand side is exactly the V-shaped formula. The function being minimized in this procedure is the same as in the previous method

$$\sum_{i=1}^{n} w_i \left(y(x_i) - y_i\right)^2 \tag{23}$$

where weights are $w_i = \frac{1}{1+(x_i-x^*)^2}$ just as for the V-shaped method.

This method requires at least four strikes per maturity, since we are fitting four parameters. If fewer strikes are given we recommend using other methods instead.

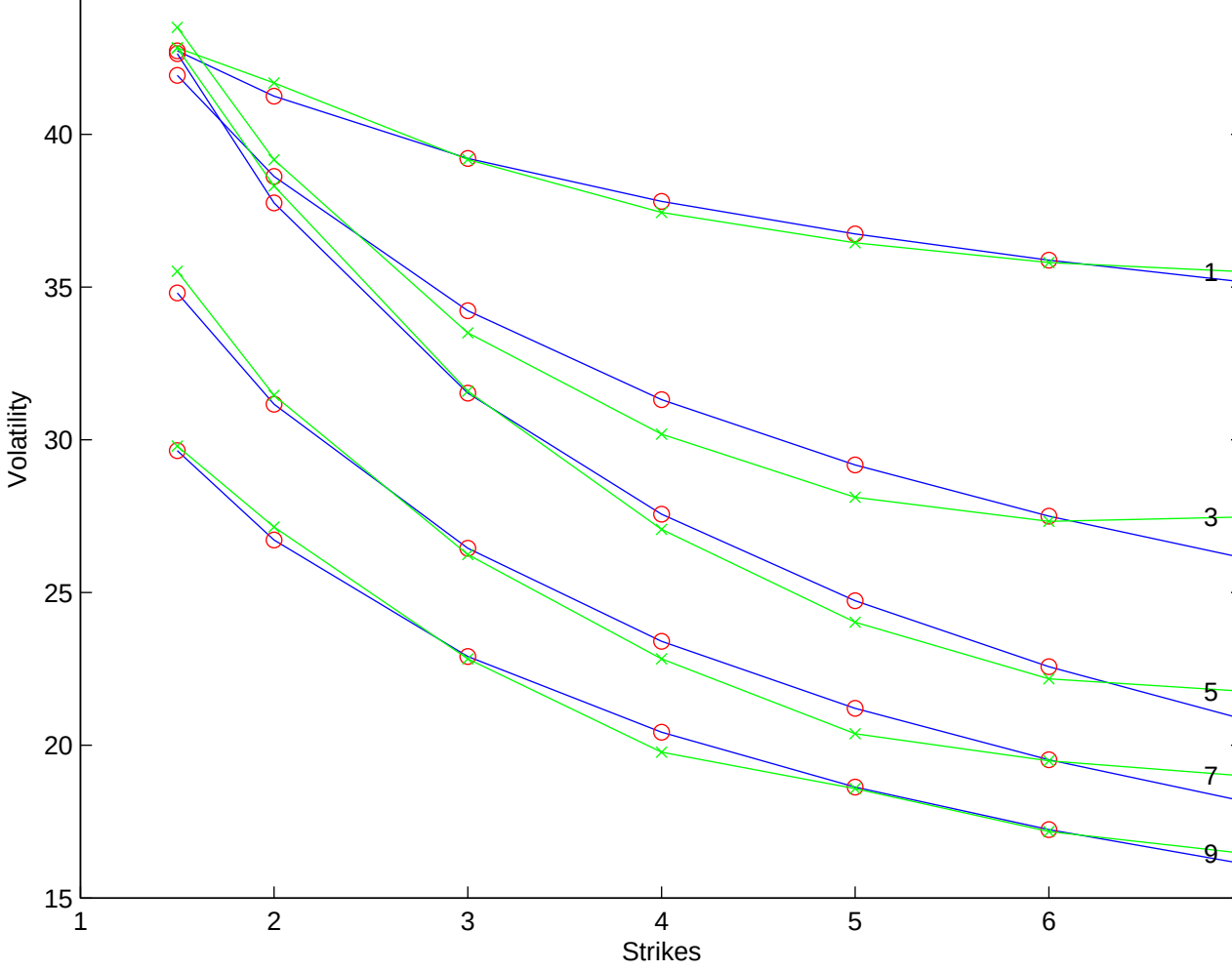

Figure 6: CEV method. Blue lines are fitted vols and green lines are the original ones.

5.4 CEV Method.

CEV method is a stochastic interpolation method using CEV model to model Libor forward rates as described in [3]. The way the model was used though is through asymptotic approximation formulas as described in [1]. We refer an interested reader to these papers for exact formulas. We assume we are given a vector of strikes $\{K_i\}_{i=1}^n$ and a vector of Black volatilities $\{\sigma_i\}_{i=1}^n$. The fit is done again in the sense of the nonlinear least squares. The model is described by the SDE

$$dF_t = \alpha F_t^\beta dW_t, \tag{24}$$

and, hence, has two parameters (α, β). The function being minimized in this procedure is

$$\sum_{i=1}^{n} w_i \left(blackToNormal(\sigma_i, K_i) - cevToNormal(\alpha, \beta, K_i)\right)^2 \tag{25}$$

where $blackToNormal$ and $cevToNormal$ are conversion functions from [1], and the weight function is $w_i = \frac{1}{\sqrt{1+(K_i-F)^2}}$. CEV model can be viewed as a special case of the SABR model discussed next.

This method requires at least two strikes per maturity, since we are fitting two parameters. If fewer strikes are given we recommend using other methods instead.

5.5 SABR Method.

SABR method is a stochastic interpolation method using SABR model to model Libor forward rates as described in [3]. The way the model was used though is through asymptotic approximation formulas as described in [1]. We refer an interested reader to these papers for exact formulas. We assume we are given a vector of strikes $\{K_i\}_{i=1}^n$ and a vector of Black volatilities $\{\sigma_i\}_{i=1}^n$. The fit is done again in the sense of the nonlinear least squares. The model is described by the SDE

$$\begin{cases} dF_t & = \sigma_t F_t^\beta dW_t^1, \\ d\sigma_t & = \nu \sigma_t dW_t^2, \\ < dW_t^1, dW_t^2 > & = \rho dt, \\ \sigma_0 & = \alpha \end{cases}$$

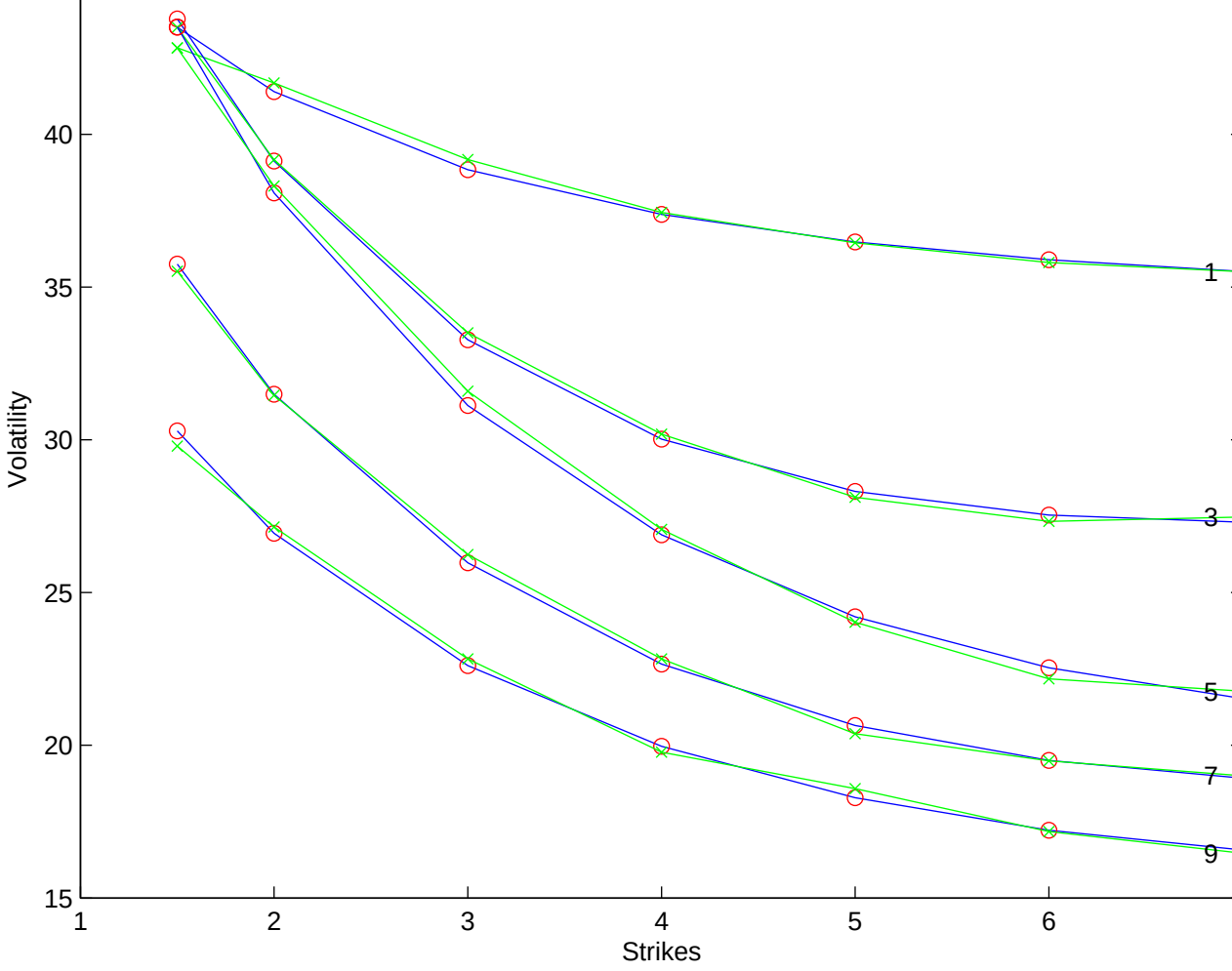

Figure 7: SABR method. Blue lines are fitted vols and green lines are the original ones.

and, hence, has four parameters $(\alpha, \beta, \rho, \nu)$. The function being minimized in this procedure is

$$\sum_{i=1}^{n} w_i \left(blackToNormal(\sigma_i, K_i) - sabrToNormal(\alpha, \beta, \rho, \nu, K_i) \right)^2 \qquad (26)$$

where $blackToNormal$ and $sabrToNormal$ are conversion functions from [1], and the weight function is $w_i = \frac{1}{\sqrt{1+(K_i-F)^2}}$. The calibration is done in two stages. First, we fit CEV, which can be viewed as SABR with $\nu = 0$. Then we fit SABR using β parameter obtained in the previous stage as SABR's initial guess.

This method requires at least three strikes per maturity, since we are fitting three parameters. If fewer strikes are given we recommend using other methods instead.

6 Volatility Cube Creation I.

To construct the volatility cube we go through the following steps. First we strip cap volatilities creating maturity interpolators along the way. Then we add an ATM caplet column to the swaption matrix. To do this we compute the forward Libor rate for each of the swaption option terms, and then interpolate on the swaption option terms and these forward rates to get the corresponding volatilities. We usually interpolate first along the term direction using maturity interpolators, and then do smile interpolation.

Then we change swaption option terms to be equally spaced grid going from cap frequency to the highest swaption option term with cap frequency step. We interpolate linearly to fill out the missing cells. This is done because we want to be able to recover cap volatilities exactly without the need to fit smile interpolators in the getVol function. Then for each of the grid nodes, we fit smile interpolator to the caplet volatilities.

Next, we fill out the whole swaption matrix with smile interpolators using these initial ones that we have in the caplet ATM column adjusting one of the parameters to match the swaption ATM volatility. For each of the smile interpolator type it is done a little differently.

- For piece-wise linear interpolator, we simply scale of the values by a constant to match the right swaption ATM volatility.

- For V-shaped and hyperbolic interpolators, we change y^* to match the right swaption ATM volatility.
- For CEV and SABR interpolators, we change α to match the right swaption ATM volatility.

The third one involves a one dimensional root solving. Lastly, we allow a user to provide his parameter changes and adjust our parameters according to these changes. Now, we have all the components to interpolate in the cube.

7 Volatility Cube Creation II.

In some cases, cap volatilities may have been specified not on a uniform grid of strikes vs. maturities, but on a nonuniform grid, when each maturity has different set of strikes. Such situation, for example, may arise when cap volatilities are specified for relative strikes in terms of moneyness. In this case our stripping procedure will not work, since we could only strip before smile fitting when we had all the cap volatilities for a given strike. Now, we need to combine stripping and smile fitting in the stripping step. The algorithm is as follows. For each cap maturity, for each strike at this maturity, we get all the caplet volatilities for this strikes for all the preceding maturities using already constructed smile interpolators for these maturities. Then we compute all the forward cap values for each strike by subtracting cap value for this strike and last maturity (computed using the caplet volatilities for preceding maturities we obtained on the previous step) from the current cap value for this strike (computed using the given cap volatility for this strike and maturity). Next, we fit a smile interpolator to these forward cap values, and add the resulting smile interpolator to our collection of smile interpolators. This concludes the step. Across maturity, we assume constant caplet volatilities for now, although this assumption may be relaxed in the future versions. After obtaining all the smile interpolators for all the cap maturities, we use piece-wise constant interpolation across maturity (for consistency with stripping method) to fill the smile interpolators for all the swaption option terms, and proceed the same way as in the previous section.

8 getVol Function.

The signature of the getVol method of the volatility cube class is

`double getVol(double optionTerm, double swapTerm, double strike).`

We will use the following notations: T_i^o for the i-th option term, where $i = 1, \ldots, N^o$, T_i^s for the i-th swap term, where $i = 1, \ldots, N^s$, and K_i for the i-th strike, where $i = 1, \ldots, N^k$.

The algorithm consists of several steps. First, using binary search algorithm, we find the corners of the cell in the swaption volatility grid containing given point (T^o, T^s). Denote them T_L^o, T_R^o, T_L^s, and T_R^s respectively.

If $T^o \leq T_1^o$ then $T_L^o = T_R^o = T_1^o$.

If $T^o > T_{N^o}^o$ then $T_L^o = T_R^o = T_{N^o}^o$.

If $T_1^o < T^o \leq T_{N^o}^o$ then $T_L^o = T_{I^o}^o$ and $T_R^o = T_{I^o+1}^o$ such that $T_L^o < T^o \leq T_R^o$.

Similarly,

If $T^s \leq T_1^s$ then $T_L^s = T_R^s = T_1^s$.

If $T^s > T_{N^s}^s$ then $T_L^s = T_R^s = T_{N^s}^s$.

If $T_1^s < T^s \leq T_{N^s}^s$ then $T_L^s = T_{I^s}^s$ and $T_R^s = T_{I^s+1}^s$ such that $T_L^s < T^s \leq T_R^s$.

Next, we get interpolated volatility along the strike direction for each corner of the rectangle with vertices, and do a bilinear (or linear, if $T_L^o = T_R^o$ or $T_L^s = T_R^s$) interpolation with option direction first.

The function that interpolates along the strike direction has the following signature

`double clippedInter(int optionIndex, int swapIndex, double strike).`

First, we clip the strike, i.e., if $K > upper_{I^o,I^s}$ we set $K = upper_{I^o,I^s}$. Similarly, if $K < lower_{I^o,I^s}$ we set $K = lower_{I^o,I^s}$. Here *upper* and *lower* are two dimensional arrays of upper and lower boundaries respectively of the strike range on which we do smile interpolation/extrapolation. On the rest of the strike values we just perform flat extrapolation. Now, we simply invoke $StrikeInterpolators(I^o, I^s)$'s interpolation method with the argument K. Lastly, we use the four volatilities we just obtained (or 2 or 1, depending on the position of (T^o, T^s) on the swaption volatility grid) to perform linear interpolation and return the resulting value. Formula for linear interpolation is

$$y = \frac{(x_{i+1} - x)y_i + (x - x_i)y_{i+1}}{x_{i+1} - x_i}, \tag{27}$$

where $x_i < x \leq x_{i+1}$.

9 Test Results.

We have tested our methodology for four major currencies: USD, GBP, EUR, and JPY. For each date the data consisted of a yield curve, a cap volatility matrix, a swaption volatility matrix, and EDFO prices or volatilities.

- A yield curve consisted of terms starting at one day and ending at 30 years. It was stripped from cash rates under 2 years, and swap rates from 2 years on provided by Bloomberg function **SWDF**.

- A cap volatility matrix was a matrix of volatilities for given pairs of absolute strikes and cap maturities. The number of maturities was usually ranging from 10 to 20 with maturities ranging from 1 year up to 10 or 20 years. The number of strikes ranged from six to twelve. The ATM volatilities were not included at the current stage. The provider we used was **ICAP**, and the Bloomberg commands to get the corresponding volatility matrices for the currencies above are **ICAU**, **ICAB**, **ICAE**, and **ICAJ** respectively.

- A swaption volatility matrix was a matrix of the ATM European swaption volatilities for given pairs of option and swap terms. The swap terms ranged from 1 to 30 years, and the option terms ranged from 1 month to 10 years. The volatility taken was the average of bid and ask volatilities. The provider we used was **Tullet & Tokyo**. The way to obtain these matrices is to start with Bloomberg command **MRKT**, and then follow the instructions.

- EDFOs come in two different forms. The historical data is stored in terms of prices, when the current data is the implied volatilities. The ticker consists of four or five characters: first two identify the EDFO's code for particular country(**ED** for **USD**, **ER** for **EUR**, **YE** for **JPY** and **L** for **GBP**), the third character identifies the month (**H** for **March**, **M** for **June**, **U** for **September**, and **Z** for **December**), and the rest identify the year (last digit for current and future ones and last two digits for the years in the past).

Table 1 provides timings of volatility cube construction in milliseconds for different stripping methods and different smile interpolation methods. Since the construction consists of stripping and filling the cube with smile interpolators, these parts can be timed separately.

Table 1: Volatility cube construction timings

Construction Part	Method	Time
Stripping	Piece-wise Constant	13
Stripping	Piece-wise Linear	400
Stripping	Piece-wise Quadratic	1410
Smile Interpolation	Piece-wise Linear	15
Smile Interpolation	V-Shaped	63
Smile Interpolation	Hyperbolic	196
Smile Interpolation	CEV	277
Smile Interpolation	SABR	387

10 Conclusions And Recommendations.

From the tests we have conducted so far, we think the best method would be piece-wise quadratic inexact stripping with amelioration in conjunction with SABR interpolation across the strikes. Since some of the other methods like piece-wise constant stripping and piece-wise linear stripping with amelioration, for instance, and piece-wise linear smile interpolation seem to be market's standard methods, we propose that they also to be available to our clients.

11 Future Work.

One of the possible things to do in the future is to use different interpolation types in $getVol$ function across the swaption volatility grid. Instead of bilinear, we can use a biquadratic spline with amelioration, for instance. Another direction for future development can be incorporation of the relative strike space into $getVol$ function. We can do it the following way. First, for each caplet node, say (t, K), we find cap maturities T_{i-1} and T_i such that $T_{i-1} < t \leq T_i$, use smile interpolators to get volatility for $(T_{i-1}, \frac{KF_{T_{i-1}}}{F_t})$ and $(T_i, \frac{KF_{T_i}}{F_t})$, and interpolate between them linearly in time to get the volatility at (t, K). Then we compute the resulting forward cap values, and corresponding implied cap volatilities. Next, we perform a few (possibly just one, for speed considerations) iteration consisting of stripping the cap volatilities again, fitting the smile interpolators, and performing the relative strike procedure we described above. After we are satisfied with the error between

reconstructed cap volatilities and the original ones, we can interpolate in the described way on any given relative strike and maturity.

12 C++ Classes.

- **VolCube** - Volatility cube class encapsulating cap/floor volatility object and swaption volatility object, and providing an access function getVol as described above.
- **CapVol** - Cap/Floor volatility interface performing stripping and smile interpolation. It can also reconstruct the original cap volatility surface for validation reasons.
- **CapVol1** - class extending **CapVol** interface. Implements uniform strike stripping.
- **CapVol2** - class extending **CapVol** interface. Implements nonuniform strike stripping.
- **SwaptVol** - Swaption volatility class.
- **EDFVol** - Eurodollar Future options volatility class. Performs conversion of EDFOs' prices into their implied Black volatilities.
- **YieldCurve** - Yield curve class providing various curve functions such as rate, discount, forward (continuous), libor (simple), swap rate, etc. Allows to be constructed from simple and swap rates by standard stripping.
- **SwapCurveStripper** - Swap Curve stripper class. Performs stripping swap rates into discount factors for the yield curve class.
- **Conventions** - a header file with usual cap and swap date frequencies.
- **ConversionFormulas** - a collection of volatility conversion functions between Black, normal, CEV and SABR models.
- **ImplVol** - Implied volatility class. Performs conversion of prices into Black implied volatilities. Uses **bsUtils** function collection (cumulative normal, inverse cumulative normal, Black's formula, etc.), **OptData**

class for data encapsulation, and **RootFinderBrent** class for 1D root solving.

- **ForwardCapVolCalc** - Forward cap volatility calculator. Calculates forward implied cap volatility. It is very similar to the **ImplVol** class, and, therefore, will probably be eliminated in the future.
- **RootFinderBrent** - 1D Root Finder using Brent's algorithm from [5].
- **RootFinderNewton** - 1D Root Finder using global Newton's algorithm from [5].
- **MinimizerTrust** - Trust region based optimization method used in all the optimizations throughout this project. Originally provided by Arun Verma in C.
- **Eigen** - Linear algebra operations for Arun's optimization routine.
- **NLSSolver** - Non-linear least square solver class, extends **MinimizerTrust**.
- **ModelParam** - Model parameter class encapsulating parameter and boundaries vector, used in **NLSSolver**.
- **Matrix** - generic 2D array. There is also a collection of elementary matrix operations **MatrixOper**, **LU**, and **SVD** performing corresponding matrix decompositions and also applicable for a simple linear regression. In the future, I intend to unify all the linear algebra operations in the **NumMatrix** class that will extend the **Matrix** class.
- **ReportDate** - generic date class. Originally provided by Mircea Marinescu.
- **Parser** - generic tokenizer class. Originally provided by Mircea Marinescu. Alternatively, I have also used a C-style function collection **ParseUtils**.
- **StopWatch** - a class performing task timing operations.
- **Exception** - an exception class. All the error handling was done using C++ exception mechanism.

- **Cap** - Cap class. Encapsulates cap instrument and performs its pricing.
- **Swaption** - Swaption class. Encapsulates European swaption instrument and performs its pricing.
- **Stripper** - Stripper interface to all the stripping methods described in this document. It has only two methods: it runs and can return maturity interpolator when asked.
- **ConstStripper** - Constant caplet volatility stripper.
- **LinearStripper** - Linear caplet volatility stripper.
- **QuadraticStripper** - Quadratic inexact stripper.
- **QuadraticStripper2** - Quadratic exact stripper.
- **Function1D** - a 1D function interface, possibly excepting parameters, and possibly integrable. Used as a building block of all the splines.
- **BSearch** - generic binary search class implemented from [5] algorithm. It is used in all the splines at interpolation stage.
- **MaturInter** - Maturity interpolator class. It has a collection of pointers to **Function1D** objects, and a binary search pointer for effective searches.
- **ConstFunc** - Constant 1D functional object. Used in the piece-wise constant caplet volatility stripping.
- **LinearFunc** - Linear 1D functional object. Used in the piece-wise linear caplet volatility stripping. It is parameterized by the end points values.
- **QuadFunc** - Quadratic 1D functional object. Used in the piece-wise quadratic caplet volatility stripping. It is parameterized by both coefficients and end points plus the mid point values.
- **StrikeInter** - Strike interpolator interface to all the smile interpolators described in this document. It is also used to set the space flags (strike and volatility).

- **PWLinearInter** - Piece-wise linear smile interpolator.
- **VShapedInter** - V-Shaped smile interpolator.
- **HyperbInter** - Hyperbolic smile interpolator.
- **CEVInter** - CEV smile interpolator.
- **SABRInter** - SABR smile interpolator.
- **SmileCalibrator** - a class performing smile fitting. For a nonuniform strike case, I also have another version called **SmileCalibrator2**.
- **CEVAdjuster** - class performing a 1D root solving to find an appropriate ATM adjuster of caplet CEV smile interpolators to fit a swaption ATM vol.
- **SABRAdjuster** - class performing a 1D root solving to find an appropriate ATM adjuster of caplet SABR smile interpolators to fit a swaption ATM vol.
- **ValFactory** - an interface to all the value factories that take flat forward values and return node values. It is used in both yield curve construction and initial guess for linear and quadratic stripping. There are three class derived from this interface **ValFactory1**, **ValFactory2**, and **ValFactory3** respectively. The first and the third are different flavors of linear values factory, and the second one is used for the quadratic ones.

13 Prototype And Pictures.

We have developed a C++ prototype with Matlab interface that allows a user to enter business date, currency, choose the interpolation types, volatility and strike spaces, and indicate whether to use EDFOs or not, and plot the interpolated caplet volatility surface or smile fits for chosen option terms. Below are some pictures from some spot tests performed using this interface. They illustrate the ideas conveyed in this document.

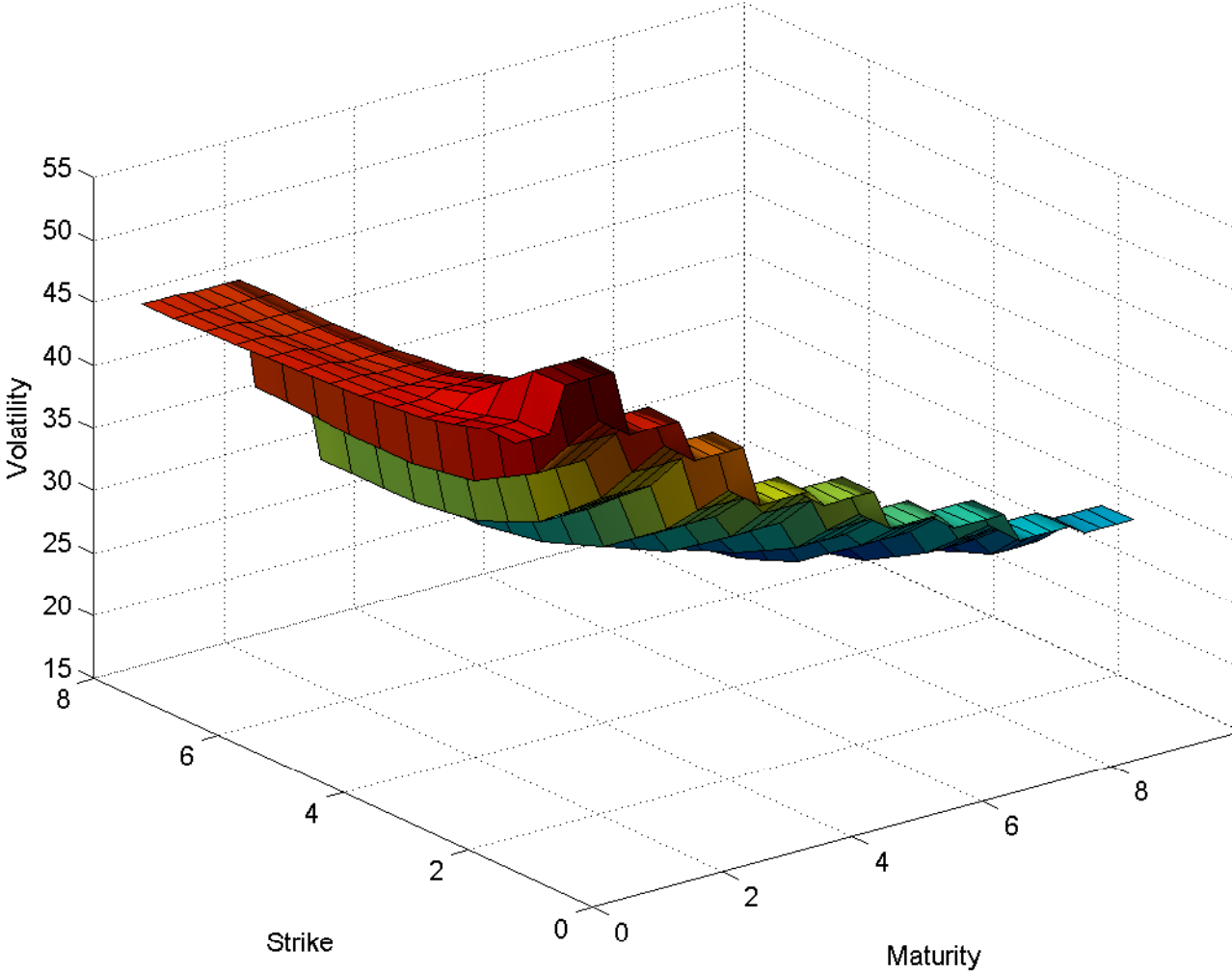

Figure 8: Piece-wise constant stripping with piece-wise linear smile.

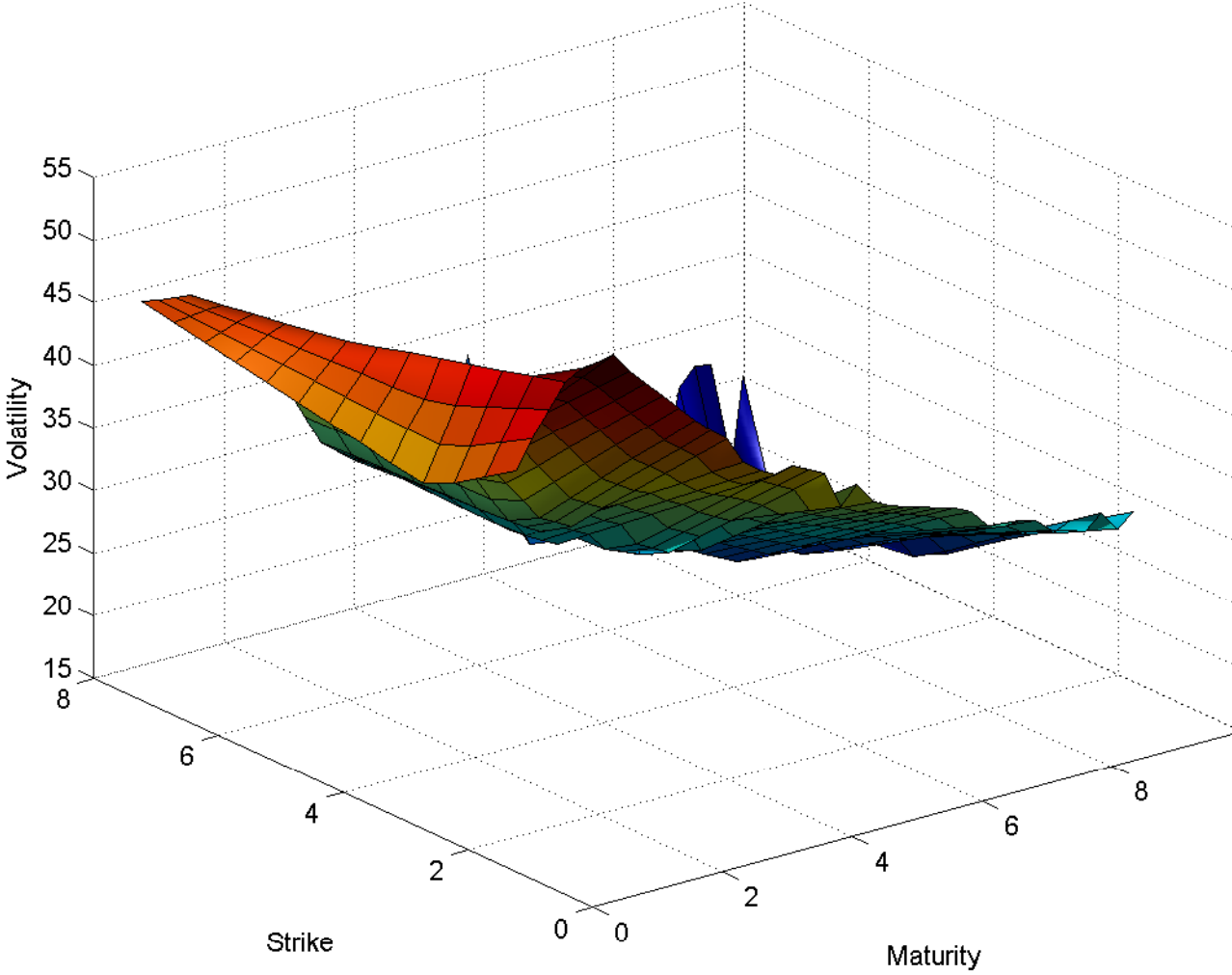

Figure 9: Piece-wise linear stripping with V-shaped smile.

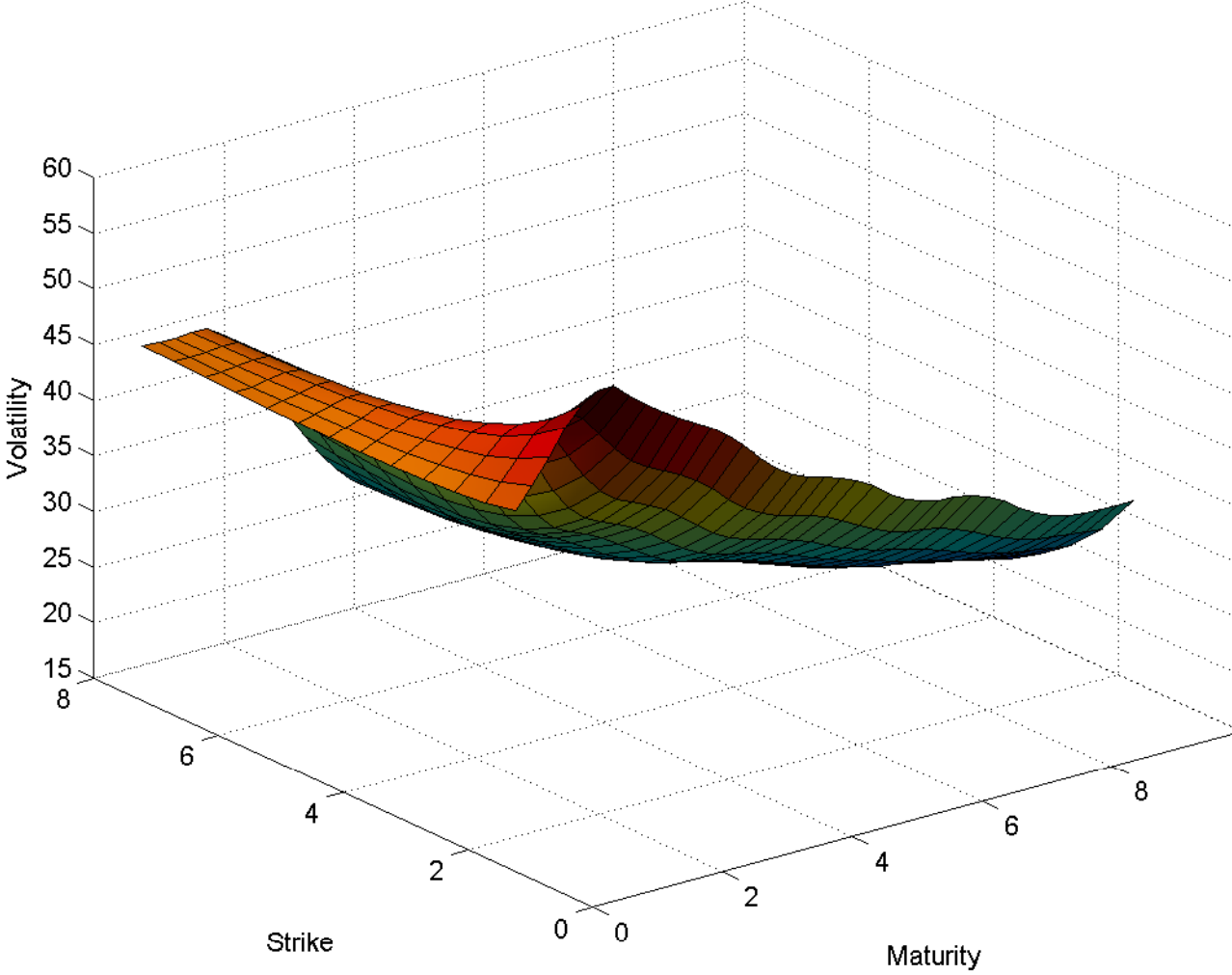

Figure 10: Piece-wise quadratic stripping with SABR smile.

References

[1] Pat Hagan, "Equivalent Volatilites For Determinstic Vol Formulas", *Bloomberg Technical Report.*

[2] Pat Hagan, "Volatility Cube", *Bloomberg Technical Report.*

[3] Patrick S. Hagan, Deep Kumar, Andrew S. Lesniewski, and Diana E. Woodward. "Managing smile risk", Wilmott, 2001.

[4] Pat Hagan, Michael Konikov, "Interpolation Methods", *Bloomberg Technical Report.*

[5] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, "Numerical Recipes in C : The Art of Scientific Computing", 2nd edition, May 18, (2004).

[6] Graeme West, "The SABR Model For Equity Derivative Smiles", Financial Modelling Agency

bloomberg.com

Frankfurt
+49 69 92041 0

London
+44 20 7330 7500

San Francisco
+1 415 912 2960

Singapore
+65 6212 1000

Tokyo
+81 3 3201 8900

Hong Kong
+852 2977 6000

New York
+1 212 318 2000

São Paulo
+55 11 3048 4500

Sydney
+61 2 9777 8600