Technical Articles

See All Technical Articles

Estimating Option-Implied Probability Distributions for Asset Pricing

By Ken Deeley, MathWorks

Forecasting the performance of an asset and quantifying the uncertainty associated with such a forecast is a difficult task: one that is frequently made more difficult by a shortage of observed market data.

Recently, there has been interest from central banks in using observed option price data for creating forecasts, particularly during periods of financial uncertainty [1], [2]. Call and put options on an asset are influenced by how the market believes that asset will perform in the future. This article describes a workflow in which MATLAB® is used to create a forecast for the performance of an asset, starting with relatively scarce option price data observed from the market. The main steps in this workflow are:

- Computing implied volatility from market data
- Creating additional data points using SABR interpolation
- Estimating implied probability densities
- Simulating future asset prices
- Presenting the forecast uncertainty in a fan chart

The fan chart conveys information in an accessible graphical form that enables financial professionals to communicate their forecasted projection and uncertainties to a nonspecialist audience (Figure 1).

The MATLAB code used in this article is available for download.

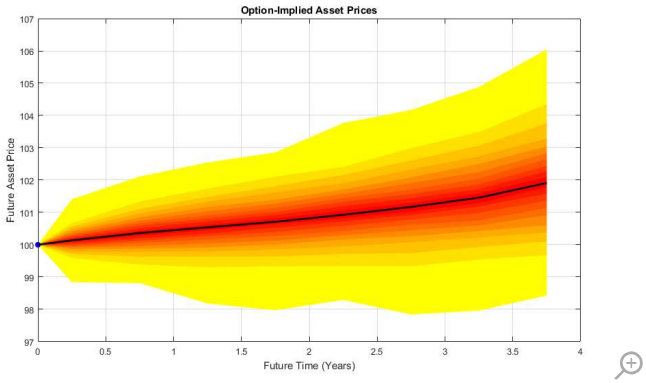

Figure 1. A fan chart representing an asset price forecast and associated uncertainty.

Computing Implied Volatility from Market Data

A key challenge facing analysts is gaining insight from a small amount of available market data. One technique for creating additional data points is interpolation in (K, σ)-space, where K is the strike price and σ is the asset volatility. To use this technique, we first must compute the implied volatility from the market data.

We assume, at a minimum, that we can observe strike-call price pairs (or strike-put price pairs) for a given asset, and that we also know the underlying asset price, risk-free rate, and expiry time for the option. We store this data in a MATLAB table using the following notation (Figure 2).

- K – strike price ($)
- C – call price ($)
- P – put price ($)
- T – expiry time of the option (years)
- rf – risk-free rate (decimal number in the range [0, 1])
- S – underlying asset price ($)

D

48x6 table

	1 K	2 C	3 P	4 T	5 rf	6 S
1	98	2.1230	0.0011	0.2500	0.0050	100
2	98.8000	1.3259	0.0018	0.2500	0.0050	100
3	99.6000	0.5652	0.0394	0.2500	0.0050	100
4	100.4000	0.0953	0.3691	0.2500	0.0050	100
5	101.2000	0.0041	1.0785	0.2500	0.0050	100
6	102	0.0013	1.8737	0.2500	0.0050	100
7	98	2.3683	0.0011	0.7500	0.0050	100
8	98.8000	1.5831	0.0145	0.7500	0.0050	

Figure 2. Observed market data in a MATLAB table. This hypothetical data has six observations relating to options with a given expiry time and eight distinct expiry times, giving 48 observations in total.

When we are interpolating in (K, σ)-space, the asset volatility, σ, is measured as a decimal number in the range [0, 1]. We begin by analyzing the call price data separately by computing the Black-Scholes implied volatilities using the

Financial Toolbox™ function `blsimpv`:

```
D.sigmaCall = blsimpv(D.S, D.K, D.rf, D.T, D.C, [], [], [], {'call'});
```

A plot of the results shows that for this data set, the highest volatilities are associated with the end points of the data corresponding to expiry time T=0.25 (Figure 3). The plot also shows that volatility increases as the strike price moves away from some central value, which appears to be close to the underlying asset price S = 100.

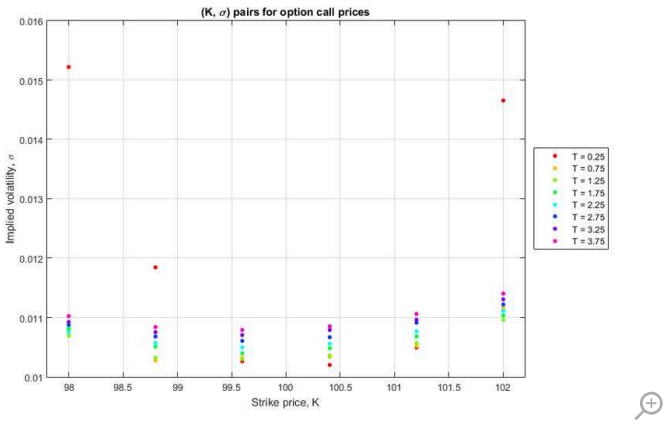

Figure 3. Observed market data in strike-volatility space, stratified by the option expiry time.

Creating Additional Data Points

Several approaches can be used to generate additional points in (K, σ)-space. Simpler methods include fitting quadratic functions to the data for each expiry time, or using the `interp1` function to construct cubic spline interpolants. We choose to use SABR interpolation to create additional data points, since this technique can often give more accurate results at the endpoints of the data set.

The SABR model is a four-parameter stochastic volatility model [3] used by financial professionals to fit *volatility smiles*, named for the shape of the resulting curves. Fitting the volatility smile is a two-step process in MATLAB. First, we calibrate the SABR model using the `lsqnonlin` solver from Optimization Toolbox™. This calibration minimizes the norm of the difference between the observed data and the candidate SABR smile, resulting in a vector of optimal parameters for the SABR model. Second, we use the optimal parameters together with the Financial Instruments Toolbox™ function `blackvolbysabr` to interpolate across the required range of strike prices.

The SABR technique works well even for fitting the awkward data corresponding to the expiry time T = 0.25 (Figure 4).

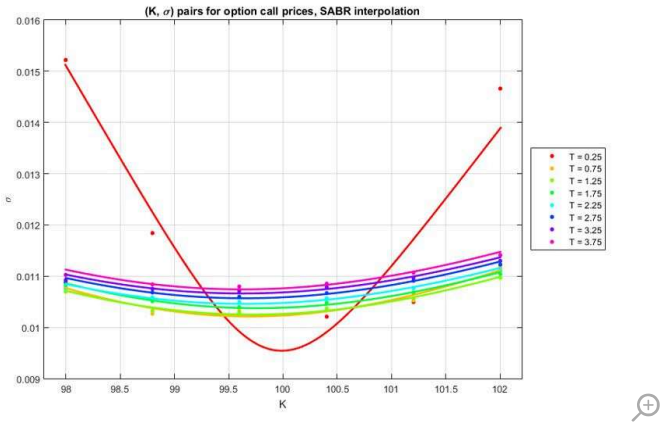

Figure 4. Volatility smiles for each expiry time obtained using SABR interpolation.

Estimating Implied Probability Densities

After interpolation in (K, σ)-space, we obtain enough data points to estimate the implied strike price density functions at each expiry time. To do this we use a computational finance principle developed by Breeden and Litzenberger [4], which states that the probability density function f(K) of the value of an asset at time T is proportional to the second partial derivative of the asset call price C = C(K).

We first transform the data to the original domain ((K, C)-space) for each expiry time using the `blsprice` function:

```
T0 = unique(D.T);
S = D.S(1);
rf = D.rf(1);
for k = 1:numel(T0)
    newC(:, k) = blsprice(S, fineK, rf, T0(k), sigmaCallSABR(:, k));
end
```

Here, `fineK` is a vector defining the range of strike prices used for the interpolation and `sigmaCallSABR` is the matrix created using SABR interpolation in which the columns contain the interpolated volatility smiles for each expiry time.`

We then compute the numerical partial derivatives with respect to the strike price. This can be done efficiently in MATLAB using the `diff` function.

```
dK = diff(sampleK);
Cdash = diff(newC) ./ repmat(dK, 1, size(newC, 2));
Cddash = diff(Cdash) ./ repmat(d2K, 1, size(Cdash, 2));
```

We also use logical indexing to remove spurious negative values that appear as unwanted artifacts of this process. The resulting curves for each expiry time show that as the expiry time increases, the functions become less complete (Figure 5). We will need to extrapolate these functions before we can use them to estimate implied probability densities, since they do not define complete functions over the range of strike prices of interest.

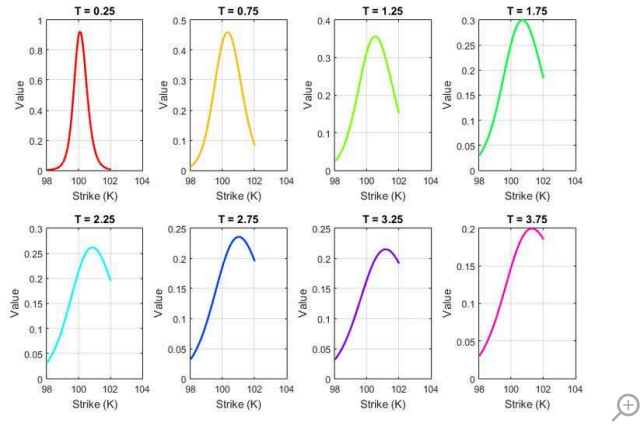

Figure 5. Option-implied functions approximated from the interpolated market data.

To extend the definitions of these functions over the complete strike price domain we create a linear interpolant using the `fit` function from Curve-Fitting Toolbox™. It is then easy to extrapolate from this interpolant to cover the range of interest.

```
for k = 1:numel(T0)
    pdfFitsCall{k} = fit(pdfK, approxCallPDFs(:, k), 'linear');
end
```

Here, `pdfK` is a vector defining the required range of strike prices and `approxCallPDFs` is a matrix storing the approximations to the implied densities for each expiry time in its columns. By extrapolating and normalizing the area under each function to ensure that we have valid probability density functions, we obtain the implied densities (Figure 6). Note that the distribution modes gradually move upwards as the time to expiry increases, and there appears to be a trend towards increasing volatility over time.

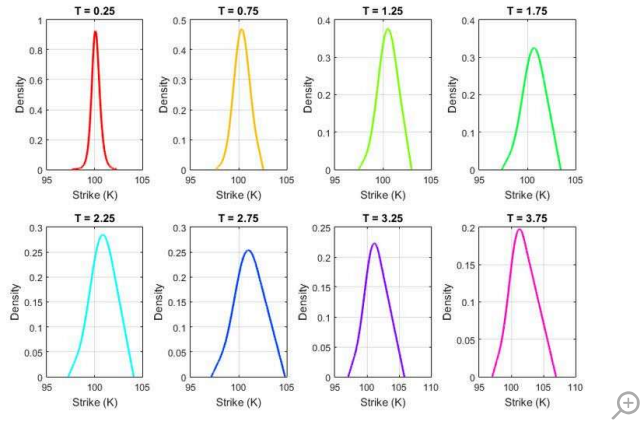

Figure 6. Final approximations to the implied density functions, after linear extrapolation and normalization.

Simulating Future Asset Prices

Now that we have complete probability distributions for the asset prices at all future times we can randomly sample from each distribution to create a forecast matrix. We need to randomly sample (with replacement) from a given

range of asset prices according to the probability distributions defined in the previous step. This simulation is straightforward to implement using the `randsample` function from Statistics and Machine Learning Toolbox™.

```
nSamples = 1e3;
priceSimCall = NaN(numel(T0), nSamples);
for k = 1:numel(T0)
    priceSimCall(k, :) =
         randsample(fitKCall{k}, nSamples, true, fitValsCall{k});
end
```

The MATLAB code above preallocates a matrix named `priceSimCall` in which each row represents a future time and each column represents a random asset price drawn from the corresponding probability distribution. The loop iterates over each future time, and each iteration creates a row of the `priceSimCall` matrix by randomly sampling from the appropriate distribution. The first input argument to `randsample` is `fitKCall{k}`, a vector that contains the asset strike prices from which to draw the random samples. The second input argument, `nSamples`, is the number of samples required, and the third (`true`) specifies that we want to sample with replacement. The fourth input argument `fitValsCall{k}` is the probability distribution used by the `randsample` function when drawing the random samples. Note that the independent computations performed in this application using sequential `for`-loops can be parallelized using the `parfor` construct. For larger data sets, parallelization allows the application to scale readily as long as multiple processing cores are available for computation.

Creating a Fan Chart

Fan charts are commonly used to plot the projected future evolution of a time series together with the uncertainties associated with the forecasted values. Financial Toolbox provides a `fanplot` function for creating fan charts. This function requires two inputs: the historical data and the forecast data. We define the historical data as simply a point at time T = 0 with corresponding asset price S = 100, although if we had historical data for the asset prices then we could incorporate them here.

```
historical = [0, S];
```

The forecast data comprises the expiry times together with the results from the simulation performed in the previous step.

```
forecastCall = [T0, priceSimCall];
```

We then create the fan chart with a call to the `fanplot` function.

```
fanplot(historical, forecastCall)
```

The central line in the fan chart represents the median trajectory, and the edges of each band represent different quantiles of the forecast simulation matrix (Figure 7), with bands closest to the central line indicating the more likely outcomes. In our example, there is an upward trend in the projected asset price, and as we would expect, the forecast uncertainty increases with time.

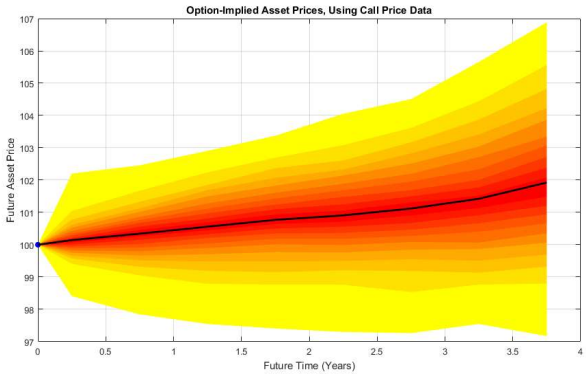

Figure 7. Fan chart showing the results of the simulation.

Assumptions and Further Improvements

To improve the accuracy of the final forecast, we could incorporate more data points. We could also modify the workflow to fit probability distributions rather than using extrapolation to complete the definition of the implied density functions, although this could introduce unwanted distributional assumptions. It is a good idea to repeat the process using the put price data. Averaging the forecasts created from the call and put price data results in the fan chart shown in Figure 1. Such averaging tends to smooth out small fluctuations, and counter bias caused by working with only one type of price data.

Note that the entire workflow can be automated using a MATLAB script that computes implied volatility, creates additional data points through interpolation, estimates the implied probability densities, and simulates future asset prices before generating a fan chart to show the results.

Published 2015 - 92950v00

References

1. “Option-implied probability distributions for future inflation,” Smith, T., *Quarterly Bulletin of the Bank of England*, 2012 Q3
2. “The Information Content of Option Prices During the Financial Crisis,” *ECB Monthly Bulletin*, February 2011
3. “Managing Smile Risk”, Hagan, P. S. et al, *WILMOTT Magazine*
4. “Prices of state-contingent claims implicit in options prices,” Breeden, D. T. and Litzenberger, R. H. (1978), *Journal of Business*, Vol. 51, pages 621–51.

Products Used

- MATLAB
- Curve Fitting Toolbox
- Financial Instruments Toolbox
- Financial Toolbox
- Optimization Toolbox
- Statistics and Machine Learning Toolbox

Learn More

- Estimating Option-Implied Probability Distributions for Asset Pricing (download)
- Calibrate the SABR Model
- Price a Swaption Using the SABR Model

View Articles for Related Capabilities

- Mathematical Modeling
- Data Analysis

View Articles for Related Industries

- Broker Dealer
- Asset Management
- Financial Services