Implementation and testing of the implied volatility surface FE5116 project

Fabio Cannizzo

March 26, 2022

Abstract

This project is a real case study, which goes through the various steps necessary to produce derivatives pricing models. It starts with the definition of a problem, a mathematical modeling choice, the implementation of a numerical framework and the calibration to market data. I already carried out the mathematical modelling part for you.

1 Problem statement

In FX markets the volatility surface is often described via a finite number of observations of vanillas European options (call and puts) at different strikes and tenors expressed in delta conventions. Our goal is

- translate the observations into pairs of (strikes, volatilities);
- interpolate the marks so that we can then price European vanilla options with any strike and maturity;
- implement a numerical pricing framework to compute the price of any generic European payoff (not just vanilla call and put options).

1.1 Payoff description

All payoffs used here are European derivatives of the currency spot price S_t observed at some time T

$$payoff = f(S_T)$$

1.2 Model choice

We assume that interest rates are deterministic. We do not make any particular assumption about the price dynamic of the currency price.

$$\begin{aligned} \frac{dM_t}{M_t} &= r(t)\,dt \quad M_t: \text{ domestic money market account} \\ \frac{dN_t}{N_t} &= y(t)\,dt \quad N_t: \text{ foreign money market account} \end{aligned} \tag{1}$$

where the deterministic functions $r(t)$, $y(t)$ are the domestic interest rate and the foreign interest rate.

1.3 Forward spot price

For a currency the spot price is defined as the price at which we agree now to exchange money (domestic currency against foreign currency) in 2 business days[1].

This same concept applies also to European derivatives of the spot price, which typically are settled 2 business days after the expiry of the option.

We define the cash exchange rate X_t as the theoretical exchange rate at which we would now (today) buy (or sell) the currency if it was possible to settle the transaction immediately. Note that this is not a market observable.

So effectively the spot price S_t is effectively the 2-days[2] forward price of the cash rate X_t. Since the forward price of X_t for maturity T given the filtration $\mathcal{F}_t$ is

$$F_t(T) = \mathbb{E}\left[X_T|\mathcal{F}_t\right] = X_t e^{\int_t^T [r(u)-y(u)]\,du} \tag{2}$$

defining the spot settlement lag as $\tau = \frac{2}{365}$ we have

$$S_t = F_t(t+\tau) = X_t\, e^{\int_t^{t+\tau} [r(u)-y(u)]\,du} \tag{3}$$

For simplicity of notation, to avoid carrying the term τ, we define

$$G_t(T) = F_t(T+\tau) = X_t\, e^{\int_t^{T+\tau} [r(u)-y(u)]\,du} \tag{4}$$

With this notation the spot price is the same as $S_t = G_t(t)$.

1.4 Implied volatility

A European call option on the spot price S_t with maturity T and strike K generate a cash flow at time $T+\tau$

$$CashFlow(T+\tau) = \max[S_T - K; 0] = \max[G_T(T) - K; 0]$$

The T-forward price of the option $C(K,T)$, expressed in terms of the current forward price $G_0(T)$ is given by the following formula:

$$\begin{aligned} C(K,T) &= G_0(T)\,N(d_1) - K\,N(d_2) \\ d1 &= \frac{\ln G_0(T) - \ln K}{\sigma\sqrt{T}} + 0.5\sigma\sqrt{T} \\ d_2 &= d_1 - \sigma\sqrt{T} \end{aligned} \tag{5}$$

where σ is the implied volatility of volatility of the option. This formula is derived under the assumption that σ is constant, however in the market we observe that it is not the case, as options with different strike and maturity have different implied volatilities. So effectively σ is a function of K and T: $\sigma(K,T)$. This is not an inconsistency, because the concept of implied volatility is purely a market convention: it is defined as the volatility which if used in the Black formula gives us the price of the option. The fact that the Black formula happens to be derived under the assumption that σ is constant is irrelevant for the purpose of this definition.

Note that for $K = 0$ the formula is not defined because $\ln(0)$ does not exist, but the price can still be obtained as a limit

$$\lim_{K\to 0^+} C(K,T) = G_0(T)$$

[1] The settlement lag is dependent on the currency pair, in most cases it is 2 business days.

[2] For simplicity here we assume that all days are business days, so that we do not have to bother counting holidays and weekends.

1.5 Probability distribution of S_T

Let $\phi(x)$ and $\Phi(x)$ be respectively the risk neutral probability density (pdf) and cumulative probability (cdf) of the spot price process S_t ate time T. The forward price of a call option on the spot price S_t with maturity T and strike K can be written as:

$$C(K) = \mathbb{E}\left[\max[S_T - K, 0] | \mathcal{F}_0\right] = \int_K^\infty (x - K)\, \phi(x)\, dx$$

Differentiating this expression twice with respect to the strike K gives the relation between the cumulative distribution function and the call price function

$$\frac{dC(K)}{dK} = \Phi(K) - 1 \tag{6}$$

$$\frac{d^2C(K)}{dK^2} = \phi(K) \tag{7}$$

From these equations, if we have a way to compute and differentiate the call price function, we can compute the cdf and the pdf of S_T.

1.6 Moneyness

We define as moneyness of an option with maturity T the relative position of the strike K with respect to the forward spot for maturity T, i.e.

$$M(K, T) = \frac{K}{G_0(T)} \tag{8}$$

1.7 No arbitrage constraints

Regardless of the model used, there are some simple no-arbitrage relationships which the forward price of a Euroepan call $C(K, T)$ option must abide to. If they are violated it means there are arbitrages in the market which can be exploited via simple trading strategies to lock in a profit.

- The price of a call option must be positive: i.e. for any K it must be $C(K) > 0$
- The price of a call option must be monotonically decreasing in strike, but it must not decrease too fast: for any consecutive pair of strikes (K_i, K_{i+1}) the price of an undiscounted call option must be decreasing, i.e. it must be

$$-1 < \frac{C_{i+1} - C_i}{K_{i+1} - K_i} < 0, \quad i = 0 \dots N - 1$$

- The price of a call option must be convex in strike: for any consecutive triplet of strikes (K_{i-1}, K_i, K_{i+1}) the gradient of the price of an undiscounted call option must be increasing, i.e. it must be

$$\frac{C_i - C_{i-1}}{K_i - K_{i-1}} < \frac{C_{i+1} - C_i}{K_{i+1} - K_i}, \quad i = 0 \dots N - 2 \tag{9}$$

- The price of a call option must be increasing along moneyness lines: for any pair of maturities $T_1 < T_2$ and strike K, it must be

$$C(K, T_1) < C\left(K\frac{G_0(T_2)}{G_0(T_1)}, T2\right) \tag{10}$$

1.8 Data

We observe in the market the spot exchange rate S_t, the prices of domestic and foreign zero coupon bonds M_t and N_t for different maturities and the Black & Scholes implied volatilities of options with standard maturity and different deltas. For your convenience, this market data is already hard-coded in the file *getMarket.m*.

TODAY	10/02/18
SPOT	1.5123

TENOR	Date	ndays	Mt	Nt	10d-put	25d-put	50-d	25d-call	10d-call
1W	17/02/18	7	0.99962	0.99923	20.80%	20.20%	20.00%	20.40%	21.60%
2W	24/02/18	14	0.99922	0.99845	21.32%	20.71%	20.50%	20.91%	22.14%
3W	03/03/18	21	0.99882	0.99766	21.84%	21.21%	21.00%	21.42%	22.68%
1M	10/03/18	28	0.99841	0.99685	22.36%	21.72%	21.50%	21.93%	23.22%
2M	10/04/18	59	0.99655	0.99322	23.40%	22.73%	22.50%	22.95%	24.30%
3m	10/05/18	89	0.99466	0.98959	24.96%	24.24%	24.00%	24.48%	25.92%
6m	10/08/18	181	0.98866	0.97818	26.00%	25.25%	25.00%	25.50%	27.00%
1y	10/02/19	365	0.97579	0.95432	27.04%	26.26%	26.00%	26.52%	28.08%
18m	10/08/19	546	0.96138	0.92864	29.12%	28.28%	28.00%	28.56%	30.24%
2y	10/02/20	730	0.94457	0.89984	31.20%	30.30%	30.00%	30.60%	32.40%

Figure 1: Market data

1.8.1 Volatility marks

Volatility marks for are given per maturity and delta value. The concept of delta is dependent on market conventions which are specific to each currency. For this project we assume some simple conventions and define delta as the first derivative of the forward price of a European option with maturity T with respect to the spot S_0. So deltas are given by these two formulas

$$\begin{aligned} |\Delta_{call}| &= N(d_1) \\ |\Delta_{put}| &= 1 - N(d_1) = N(-d_1) \end{aligned} \tag{11}$$

where $N(x)$ is the standard normal cumulative distribution function,

$$d_1 = \frac{\ln F_T - \ln K}{\sigma\sqrt{T}} + \frac{1}{2}\sigma\sqrt{T}$$

For example, for a *25d-put* with maturity 1W we see a volatility of 20.20%. This means that for a European put option with maturity 1W, strike K_{25p} and volatility 20.20% the undiscounted delta in absolute value is 0.25 (note a factor 100 is included by convention), i.e.

$$0.25 = 1 - N(d_1) \tag{12}$$

where $\sigma = 20.20\%$, $T = 1W = \frac{7}{365}$. Note that in expression (12) everything is known except for the strike K_{25p}, which needs to be obtained via root search.

Note that for *50d* it is not specified if it is a call or a put, because they are equivalent.

1.8.2 Assumptions

For simplicity you can assume that interest rate curves and volatilities use the same tenor dates. You cannot make assumptions on the number of tenors or on the number of deltas (i.e. your program must work also with a different market, containing a different number of tenor or delta points). Extrapolation of interest rate beyond the last tenor is allowed up to 30 days. Extrapolation of volatility beyond the last tenor is not allowed. Querying the vol surface with tenor zero is allowed.

2 Tasks

This section defines a number of guided tasks which you need to carry out.

2.1 Black formula

Implement the Black formula for the forward price of a call option, as described in equation (5), using the signature below:

```
% Inputs:
%    f: forward spot for time T, i.e. E[S(T)]
%    T: time to expiry of the option
%    Ks: vector of strikes
%    Vs: vector of implied Black volatilities
% Output:
%    u: vector of call options undiscounted prices
function u = getBlackCall(f, T, Ks, Vs)
```

2.2 Interest rate interpolation

Assume that interest rates are constant within consecutive pairs of zero coupon maturities. This means that the functions $r(t)$ and $y(t)$ appearing in equations (1) are constant in each of the intervals $[T_i, T_{i+1}]$. The local rates related to each interval must be computed from discount factors using a boot-strapping technique. Assume that after the last tenor T_N the rate curve continues constant at the same value of the last interval, as specified in section 1.8.2. Implement the two functions specified below.

```
% Inputs:
%    ts: array of size N containing times to settlement in years
%    dfs: array of size N discount factors
% Output:
%    curve: a struct containing data needed by getRateIntegral
function curve = makeDepoCurve(ts, dfs)
```

```
% Inputs:
%    curve: pre-computed data about an interest rate curve
%    t: time
% Output:
%    integ: integral of the local rate function from 0 to t
function integ = getRateIntegral(curve, t)
```

The second function must compute $\int_0^t r(u)\,du$, which is one of the ingredients needed to compute discount factors. It uses information pre-computed in the first function and passed to the argument `curve`. You are free to store in the structure `curve` whatever you want. If you think it is not useful to do any pre-computation, then you can just store the arguments of the first function, `ts` and `dfs`. In fact these two function could be merged into a single one. The reason for splitting them is that the first function can be used to pre-compute and cache information useful to speed up calls to the second function. This is useful because we can design our pricing framework so that the first function is called just once, at the beginning of calculations. Then the second function, which might be invoked multiple times in the context of the numerical framework, can take advantage of some pre-computed information, thus making calculations faster.

2.3 Forward spot $G_0(T)$ $(= S_T)$

```
% Inputs:
%    domCurve: domestic IR curve data
%    forCurve: domestic IR curve data
%    spot: spot exchange rate
%    tau: lag between spot and settlement
% Output:
%    curve: a struct containing data needed by getFwdSpot
function curve = makeFwdCurve(domCurve, forCurve, spot, tau)
```

Implement a function which computes the forward spot as specified in equation (4).

```
% Inputs:
%    curve: pre-computed fwd curve data
%    T: forward spot date
% Output:
%    fwdSpot: E[S(t) | S(0)]
function fwdSpot = getFwdSpot(curve, T)
```

2.4 Conversion of deltas to strikes

Write a function that, given a volatility mark (as in figure 1) computes the associated strike via root search.

```
% Inputs:
%    fwd: forward spot for time T, i.e. E[S(T)]
%    T: time to expiry of the option
%    cp: 1 for call, -1 for put
%    sigma: implied Black volatility of the option
%    delta: delta in absolute value (e.g. 0.25)
% Output:
%    K: strike of the option
function K = getStrikeFromDelta(fwd, T, cp, sigma, delta)
```

2.5 Interpolation of implied volatility in strike direction

We want to construct a function that interpolates the smile in strike for a given tenor T. Given a vector of volatility marks $(CallOrPutFlag_i, Delta_i, \sigma_i)$, with $i = 1..N$, we convert it to a vector of pairs (K_i, σ_i) using the function developed in section 2.4.

Then we perform some basic arbitrage checks and if any is violated we raise an error. Just for the purpose of checking for arbitrage, we add a dummy strike $K_0 = 0$ to the left of the vector of strikes K_i, and we compute the forward price of a European call option C_i in correspondence of each strike, thus obtaining a vector of pairs (K_i, C_i). Now we can check the arbitrages mentioned in equation (9) (the other type of strike arbitrages usually do not require checking).

Then we construct an interpolation scheme based on a natural cubic spline[3] built on the pairs (K_i, σ_i), which we use to interpolate the function $\sigma(K)$ for any $K \in [K_1, K_N]$. Outside of the interval $[K_1, K_N]$ we design an extrapolation scheme which ensures smoothness of the function $\sigma(K)$ at the junction points K_1 and K_N and gently flatten the smile for strikes very far

[3] you can use Matlab functions

out of the interval. A possible way to achieve that is to connect to the cubic spline a hyperbolic tangent function:

$$\sigma(K) = \begin{cases} spline(K_1) + a_L \tanh(b_L (K_1 - K)), & K < K_1 \\ spline(K), & K_1 < K < K_N \\ spline(K_N) + a_R \tanh(b_R (K - K_N)), & K > K_N \end{cases}$$

where a_L, b_L, a_R and b_R are chosen so that on both the left and right side of the interval $[K_1, K_N]$:

- the first derivative of $\sigma(K)$ is continuous in $K = K_1$ and $K = K_N$ (i.e. the spline and the hyperbolic tangent function have the same first derivative). This yields the conditions

  $$a_R b_R = \sigma'(K_N)$$
  $$-a_L b_L = \sigma'(K_1)$$

  Note that, because the slopes is a cubic polynomial, its first derivative is known analytically and we do not need to use finite differences (hint: you just need to find the right Matlab function to use).

- the first derivative of $\sigma(K)$ reduces by 50% at strikes defined as:

  $$K_L = K_1 \frac{K_1}{K_2}, \quad K_R = K_N \frac{K_N}{K_{N-1}}$$

  This yields the conditions:

  $$\begin{matrix} |\sigma'(K_L)| = 0.5\,|\sigma'(K_1)| \\ |\sigma'(K_R)| = 0.5\,|\sigma'(K_N)| \end{matrix} \implies \begin{matrix} 0.5 = \tanh^2(b_R (K_R - K_N)) \\ 0.5 = \tanh^2(b_L (K_1 - K_L)) \end{matrix}$$

Implement the two functions below. The first function resolve the deltas to strikes, check for arbitrages and pre-computes all data necessary to perform the interpolation of the function $\sigma(K)$. The second function performs the actual interpolation. Note that the second function should support vectorial calls.

```matlab
% Inputs:
%    fwdCurve: forward curve data
%    T: time to expiry of the option
%    cps: vetor if 1 for call, -1 for put
%    deltas: vector of delta in absolute value (e.g. 0.25)
%    vols: vector of volatilities
% Output:
%    curve: a struct  containing  data needed in getSmileK
function [curve]=makeSmile(fwdCurve, T, cps, deltas, vols)
    % Hints:
    % 1. assert vector dimension match
    % 2. resolve strikes using deltaToStrikes
    % 3. check arbitrages
    % 4. compute spline coefficients
    % 5. compute parameters aL, bL, aR and bR
```

```matlab
% Inputs:
%    curve: pre-computed smile data
%    Ks: vetor of strikes
% Output:
%    vols: implied volatility at strikes Ks
function vols=getSmileVol(curve, Ks)
```

2.6 Construction of implied volatility surface

So far we have constructed an interpolation schemes in strike for each volatility tenor. We need a second interpolation scheme to obtain the volatility at maturities T which are not exactly the volatility tenors. To compute $\sigma(T, K)$ we use a linear in variance interpolation scheme along moneyness lines (see section 1.6), based on the following equation:

$$\sigma^2(T,K)\,T = \begin{cases} \sigma_1^2(K_1)\,T, & T \le T_1 \\ \dfrac{T_{i+1}-T}{T_{i+1}-T_i}\sigma_i^2(K_i)\,T_i + \dfrac{T-T_i}{T_{i+1}-T_i}\sigma_{i+1}^2(K_{i+1})\,T_{i+1}, & T_i < T \le T_{i+1} \\ error, & T > T_N \end{cases} \tag{13}$$

where T_i and T_{i+1} are the contiguous pair of tenors enclosing T, i.e. $T_i < T < T_{i+1}$, $\sigma_i(K)$ and $\sigma_{i+1}(K)$ are the associated smile functions, K_i and K_{i+1} are respectively the strikes for tenor T_i and T_{i+1} equivalent to K in moneyness (see equation (8))

$$\frac{K}{G_0(T)} = \frac{K_i}{G_0(T_i)} = \frac{K_{i+1}}{G_0(T_{i+1})}$$

Effectively equation (13) interpolates linearly the variance at (T_1, K_1) and the variance at (T_{i+1}, K_{i+1}), where K_i and K_{i+1} are the moneyness equivalent strikes. The formula used in $[0, T_1]$ would seem like as a special case, however that is conceptually identical to the one used in $[T_i, T_{i+1}]$ if we note that the variance in $T = 0$ is zero.

Implement the two functions below. The first function construct smiles for all tenors and pre-computes all data necessary to perform the interpolation of the function $\sigma(T, K)$ as explained above. It should also perform no-arbitrage checks described in equation (10), but for simplicity only for $K = fwd$. The second function performs the actual interpolation. Note that the second function should support vectorial calls.

```
% Inputs:
%    fwdCurve: forward curve data
%    Ts: vector of expiry times
%    cps: vetor of 1 for call, -1 for put
%    deltas: vector of delta in absolute value (e.g. 0.25)
%    vols: matrix of volatilities
% Output:
%    surface: a struct  containing  data needed in getVol
function volSurf = makeVolSurface(fwdCurve, Ts, cps, deltas, vols)
```

```
% Inputs:
%    volSurf: volatility surface data
%    T: time to expiry of the option
%    Ks: vector of strikes
% Output:
%    vols: vector of volatilities
%    fwd: forward spot price for maturity T
function [vols, fwd] = getVol(volSurf, T, Ks)
```

2.7 Compute the probability density function of S_T

Implement a function which computes the cumulative distribution function

```
% Compute the probability density function of the price S(T)
% Inputs:
%    volSurf: volatility surface data
%    T: time to expiry of the option
%    Ks: vector of strikes
% Output:
%    pdf: vector of pdf(Ks)
function pdf = getPdf(volSurf, T, Ks)
```

2.8 Compute forward prices of European options

Forward prices of European options with payoff $f(S_T)$ can be written as:

$$V = \int_0^\infty f(x)\,\phi(x)\,dx$$

Implement a function which computes prices by solving the integral. You are free to use whatever integration method you prefer, including Matlab's provided ones. The function must have the signature:

```
% Computes the price of a European payoff by integration
% Input:
%   volSurface : volatility surface data
%   T : time to maturity of the option
%   payoff: payoff function
%   ints: optional, partition of integration domain
%         e.g. [0, 3, +Inf]. Defaults to [0, +Inf]
% Output:
%   u : forward price of the option (undiscounted price)
function u = getEuropean(volSurface, T, payoff, ints)
```

The argument `ints` is useful to inform the integration function about the presence of discontinuities in the payoff, where integration methods would struggle, or to avoid wasting time integrating in regions where the integral is known to be null.

3 Instructions

You are required to:

- Implement a Matlab file (.m) for each of the function requested in section 2. Each function must have **exactly** the signature and function name specified in this document, you are not allowed to modify that. You can also create additional Matlab files (.m) as you see fit.
- The file `project.m` illustrates how the functions are to be used. Make sure that file runs without errors with your implementation.
- You can work individually or in groups of up to 5 people.
- Every group must submit:
  - the Matlab implementation files (.m)
  - a document in pdf format clearly explaining your implementation of each function (do not repeat the content of this document, focus on details specific of your own implementation) and tests (see sections 3.2 and 3.4).
  - Individual reflections (appended to the main document) (see section 3.3)

- a text file (group.txt) with the names and students IDs of each group member. This should have exactly the same format as the example below :

  A123456, John Smith
  A341238, Helen Doe

- The zip file should NOT contain any directory or sub-directory. All files should be in the same directory.
- Use a zip file. Do NOT use other compression formats (e.g. rar).
- All the above material should be bundled together and submitted via *Luminus* as a single zip file.
- Warning: any deviation from the instructions above will result in mark penalties.

3.1 Implementation

- Make sure your code is well commented
- Make correct use of indentation
- Assert validity of User's input. Should input be invalid display a meaningful error messages.
- You are free to use any Matlab's function
- Make sure valid corner cases are handled properly (e.g. a call option with zero strike)
- Make all `get*` functions as fast as you can. I will test their performance, both for scalar and vectorial (where supported) invocations.
- Make all your functions robust and accurate. For example `getEuropean` should give accurate results also if the payoff contains discontinuities.

3.2 Project document

- Describe clearly the implementation of each of your functions.
- Describe the content of each of the curve objects
- Illustrate with charts the output of your `get*` functions, demonstrating their output is smooth and behaves as expected. When possible shows in the same chart also analytical results or input marks.
- Describe clearly all your tests (see section 3.4).
- Maximum 2000 words

3.3 Reflections (individual)

- List what functions you implemented.
- Describe what issues you faced and how you addressed them.
- Maximum 300 words

3.4 Tests

You should write tests for all your `getXXX` functions. The goals of testing are to make sure that:

- the function is correctly implemented
- the function is robust and does not break in correspondence of a wide range of input argument
- make sure that, if in the future somebody accidentally modify the functions, this will be caught by tests.

For each `getXXX` function you need to make sure that your tests cover all of the points above, and you need to explain that.

There are various types of tests you can perform: unit tests, regression tests, property tests, qualitative tests. Some possible ideas are:

- Market data interpolation functions are constructed from a discrete set of points. If called with one of the points used to calibrate it, it must return the original point. For example, if the `getRateIntegral` function is called with any of the tenor times T_i given in the market, it should return the integral such that $\exp(-\int_0^{T_i} r(u)\,du) = M_i$
- If a function is constant in a domain $[T_i, T_{i+1}]$, it's integral should be linear.
- The derivative of a linear function is a constant, the derivative of a constant function is zero
- An integral function is continuous
- The integral of the pdf function is 1,
- The mean of the risk eutral probability (the pdf) is the forward.
- European call options forward prices must satisfy a form of call-put parity
- The plot of a spline must look qualitatively smooth
- A call option forward price obtained with the Black formula or by numerical integration must match
- A non vanilla payoff (e.g. a digital option) can be approximated with a linear combination of vanilla options
- If the market degenerates (e.g. if there is no smile), the price of some options (e.g. a digital) can be computed analitically

In the document explain how you carry out your tests. For each test you need to state:

- what is the purpose of the test?
- how does it achieve its testing goal?
- what does it do?
- what is the pass/fail criteria?
- Illustrate and comment the test results.