Implementation and testing of the implied volatility surface and numerical pricing of European options
FE5116 project

Fabio Cannizzo

31 Mar 2024

Abstract

This project is a real case study, which goes through the various steps necessary to produce derivatives pricing models. It entails of the definition of a problem, the 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 assumption about the dynamic of the currency exchange rate.

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

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 S_t 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 X_t is not directly observable in the market, only S_t is.

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 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 $\phi_{S_T}(x)$ be respectively the risk neutral probability density (pdf) of the spot price process S_t at time T. The T-forward price of a call option on the spot price S_t with maturity T and strike K (formula (5)), can be written as:

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

Differentiating twice this expression with respect to the strike K gives the relation between the probability density function and the T-forward call price function

$$\frac{d^2C(K)}{dK^2} = \phi_{S_T}(K) \tag{6}$$

So the probability density function of S_T is the second derivative of the call price function (5) with respect to the strike K.

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.

$$M(K, T) = \frac{K}{G_0(T)} \tag{7}$$

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

$$\frac{C_i - C_{i-1}}{K_i - K_{i-1}} < \frac{C_{i+1} - C_i}{K_{i+1} - K_i}, \quad i = 0 \dots N - 2 \tag{8}$$

- 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

$$C(K, T_1) < C\left(K\frac{G_0(T_2)}{G_0(T_1)}, T_2\right) \tag{9}$$

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, a market data set compatible to the one in figure 1 is already hard-coded in the file *getMarket.m* in appendix A.

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: Example of a possible market data set

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

$$\begin{aligned} |\Delta_{call}| &= N(d_1) \\ |\Delta_{put}| &= 1 - N(d_1) = N(-d_1) \end{aligned} \tag{10}$$

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.

$$0.25 = 1 - N(d_1) \tag{11}$$

where $\sigma = 20.20\%$, $T = 1W = \frac{7}{365}$. Note that in expression (11) 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.

```
% 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 (8) (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:

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

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.

```
% 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
```

```
% 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:

$$\sigma^2(T,K)\,T = \begin{cases} \sigma_1^2(K_1)\,T, & T \le T_1 \\ \dfrac{T_{i+1}-T}{T_{i+1}-T_i}\sigma_i^2(K_i)\,T_i + \dfrac{T-T_i}{T_{i+1}-T_i}\sigma_{i+1}^2(K_{i+1})\,T_{i+1}, & T_i < T \le T_{i+1} \\ \text{error}, & T > T_N \end{cases} \tag{12}$$

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 (7))

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

Effectively equation (12) 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 (9), 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 probability density function $\phi_{S_T}(x)$ as per equation (6). The second derivative of equation (5) can be computed either in closed form (remember that σ is a function of K and must be differentiated as well) or numerically via finite differences.

```
% 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:

$$V = \int_0^\infty f(x)\,\phi_{S_T}(x)\,dx$$

Implement a function which computes prices by solving the integral. The function $\phi_{S_T}(x)$ appearing in the integral is the pdf implemented in section 6. 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
%   subints: optional, partition of integration domain into sub-intervals
%         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 `subints` is useful to inform the integration function about the presence of discontinuities in the payoff, where numerical integration methods would struggle, or to avoid wasting time integrating in regions where the integral is known to be null. For example, for a vanilla European call option with strike K, we could pass $[K, +Inf]$, for an option with payoff $1 + I_{[K_1<S_T<K2]}$, we could pass $[0, K_1, K_2, +Inf]$.

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` in appendix B demonstrates how the functions are to be used. Make sure that file runs without errors with your implementation. You are not allowed to make any change to the file `project.m`.
- You can work individually or in groups of up to 5 people.
- Every group must submit via *Canvas* one single zip file containing:
  - 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 3.3).
  - You need to include in the zip file also a file with name *group.txt* (in **lowercase** letters) containing team members' students IDs. The file must contains **only** the student IDs on separate lines (note there should be a newline also after the last ID), it should NOT contain students names or headers. The student ID should be in uppercase. Example:

```
A02111111A
A02111141E
A03116141B
```

- The zip file should NOT contain any directory or sub-directory. All files should be in the same directory.
- Use a zip file format. Do NOT use other compression formats (e.g. rar, 7z, arj).
- Warning: any deviation from the instructions above will result in mark penalties.

3.1 Implementation

You should:

- 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.

You should **NOT**:

- Use the Matlab *symbolic math* package. This is a numerical methods course!
- Make changes to the file `project.m`
- Change any of the function signature given in section 2
- Invoke directly the file `getMarket.m` from any of the function you implement.
- Make assumption about the market structure based on the file `market.m`. E.g. you should not make any assumption on the number of tenors in the rate curve, or on the number of strikes in the vol surface.

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 3.3).
- Maximum 2000 words

3.3 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 call-put parity
- The plot of a spline must look qualitatively smooth
- 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 or a Call option) can be computed analytically, which can be used as a benchmark

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.

4 Marking Criteria

1. Correct submission files (-1 mark). This is a negative mark. It is subtracted from your final mark if you do not follow to the letter submission instructions given in section 3. The correction is partially automated. For example, if you name your file `team.txt` instead of `group.txt`, some of my automated routines will fail requiring manual intervention. Therefore any deviation from the instructions is penalized.
2. Check that `project.m` runs and produce the correct output (3 marks). Note that if this fails, the points 3, 4 and 5 described below will not be marked.

3. Special cases and input error handling (1 mark). Special cases (e.g. a call with zero strike) should be handled correctly. Input errors should cause an exception with a nice error message.
4. Accuracy (3 marks). I will invoke your functions with many different input parameters. For example, I will test your `getVol` function on a fine grid of strike and maturities and compare your results with mine. Another example, I will compute the price of various European payoffs on a fine grid of strikes and maturities and compare your results with mines.
5. Performance (3 marks). I will repeat the accuracy check described at the point (4) thousands of times and compare your total execution time with mine.
6. Documentation (1 mark). I will check your document is clear and explanatory.
7. Tests (1 mark). I will check your test suite is sensible and clearly described.

A getMarket.m

```
function [spot, lag, days, domdf, fordf, vols, cps, deltas] =  getMarket()
    spot = 1.5123;
    lag = 2;  % spot rule settlement lag (we assume the lag is based on calendar d
    data = [
        7       0.99962 0.99923 20.80   20.20   20.00   20.40   21.60
        14      0.99922 0.99845 21.32   20.71   20.50   20.91   22.14
        21      0.99882 0.99766 21.84   21.21   21.00   21.42   22.68
        28      0.99841 0.99685 22.36   21.72   21.50   21.93   23.22
        59      0.99655 0.99322 23.40   22.73   22.50   22.95   24.30
        90      0.99466 0.98959 24.96   24.24   24.00   24.48   25.92
        181     0.98866 0.97818 26.00   25.25   25.00   25.50   27.00
        365     0.97579 0.95432 27.04   26.26   26.00   26.52   28.08
        546     0.96138 0.92864 29.12   28.28   28.00   28.56   30.24
        730     0.94457 0.89984 31.20   30.30   30.00   30.60   32.41
        ];
    days = data(:,1);
    domdf = data(:,2);
    fordf = data(:,3);
    vols = data(:,4:end) / 100;
    cps  = [-1, -1, 1, 1, 1];
    deltas  = [0.1, 0.25, 0.5, 0.25, 0.1];
end
```

B project.m

```
function project()

    [spot, lag, days, domdfs, fordfs, vols, cps, deltas] = getMarket();

    tau = lag / 365; % spot rule lag

    % time to maturities in years
    Ts = days / 365;

    % construct market objects
    domCurve = makeDepoCurve(Ts, domdfs);
    forCurve = makeDepoCurve(Ts, fordfs);
    fwdCurve = makeFwdCurve(domCurve, forCurve, spot, tau);
    volSurface = makeVolSurface(fwdCurve, Ts, cps, deltas, vols);

    % compute a discount factor
    domRate = exp(-getRateIntegral(domCurve, 0.8))

    % compute a forward spot rate G_0(0.8)
    fwd = getFwdSpot(fwdCurve, 0.8)

    % build ans use a smile
    smile = makeSmile(fwdCurve, Ts(end), cps, deltas, vols(end,:));
    atmfvol = getSmileVol(smile, getFwdSpot(fwdCurve, Ts(end)))

    % get some vol
    [vol, f] = getVol(volSurface, 0.8, [fwd, fwd])

    % get pdf
    pdf = getPdf(volSurface, 0.8, [fwd, fwd])

    % atm european call
    payoff = @(x)max(x-fwd,0);
    u = getEuropean(volSurface, 0.8, payoff)

    % atm european put with subintervals hint
    payoff = @(x)max(fwd-x,0);
    u = getEuropean(volSurface, 0.8, payoff, [0, fwd])

    % atm european digital call spread with subintervals hint
    payoff = @(x) (x > 0.95*fwd) .* (x < 1.05*fwd);
    u = getEuropean(volSurface, 0.8, payoff, [0.95*fwd, 1.05*fwd])
end
```