SABR

Antoine Jacquier

Title:	SABR
Author:	Antoine Jacquier
Number of pages:	7
First version:	February 19, 2017
Current version:	February 19, 2017
Revision:	0.0.0

Contents

1 SABR Expansions and arbitrage

```
In [9]: import numpy as np
        import matplotlib.pyplot as plt
```

The SABR model is the most popular model on Fixed Income desks. We investigate here some of its pitfalls. The forward process $(F_t)_{t\geq 0}$ is the unique strong solution to the following stochasic differential equation:

$$\begin{aligned} dF_t &= \sigma_t F_t^{\beta}\left(\rho dW_t^1 + \sqrt{1-\rho^2}dW_t^2\right), \\ d\sigma_t &= \nu\sigma_t dW_t^2, \end{aligned}$$

with $F_0 = 1$, $\sigma_0 = \alpha$, $\rho \in [-1, 1]$ and $\beta \leq 1$. We shall denote by $I_t(x)$, the implied volatility corresponding to European options with maturity t and log strike $x = K/F_0$.

2 Computing the density

We compute the density function corresponding to a given smile, at some fixed time horizon $t > 0$.

In [15]:
```
def density(smile, t, x, eps):
    # Computes the density at a point x = log(K/S) given a smile
    # function
    w = smile(x) * t
    wp = smile(x + eps) * smile(x + eps) * t
    wm = smile(x - eps) * smile(x - eps) * t
    w1 = (wp - wm) / (2. * eps)
    w2 = (wp - 2.0 * w + wm) / (eps * eps)
    g = (1. - 0.5 * x * w1 / w) * (1. - 0.5 * x * w1 / w) - \
        0.25 * w1 * w1 * (1. / w + 0.25) + 0.5 * w2
    sqw = np.sqrt(w)
    return g * np.exp(-x - 0.5 * (-x / sqw - 0.5 * sqw) * (-x / sqw - 0.5 * sqw)) / np.sqrt(2. *


def density2(smile, smileP, smileM, t, x, eps):
    # Given three vectors of smile (evaluated at x, e+eps,
    # x-eps), computes the density at x = log(K/S)
    w = smile * smile * t
    wp = smileP * smileP * t
    wm = smileM * smileM * t
    w1 = (wp - wm) / (2. * eps)
    w2 = (wp - 2. * w + wm) / (eps * eps)
    g = (1. - 0.5 * x * w1 / w) * (1. - 0.5 * x * w1 / w) - \
        0.25 * w1 * w1 * (1. / w + 0.25) + 0.5 * w2
    sqw = np.sqrt(w)
    return g * np.exp(-0.5 * (-x / sqw - 0.5 * sqw) * (-x / sqw - 0.5 * sqw)) / np.sqrt(2. * np


def plotDensity(smile, smileP, smileM, t, eps, logStrikes, col):
    # Plots the density using the function density2 above
    mydensity = []
    for n in range(len(logStrikes)):
        mydensity.append(
            density2(smile[n], smileP[n], smileM[n], t, logStrikes[n], eps))
    plt.plot(logStrikes, mydensity, col, linewidth=2)
```

3 Obloj's Expansion

Jan Obloj proved the following expansion for the implied volatility in the SABR model for a maturity t (where $\widetilde{x} = -x = \log(F_0/K)$):

$$I_t(\widetilde{x}) = I^0(\widetilde{x})\left(1 + I^1_H(\widetilde{x})t\right),$$

where

$$I^1_H(\widetilde{x}) = \frac{(\beta-1)^2}{24}\frac{y_0^2}{(fK)^{1-\beta}} + \frac{1}{4}\frac{\rho\nu y_0\beta}{(fK)^{(1-\beta)/2}} + \frac{2-3\rho^2}{24}\nu^2.$$

At the money ($x = 0$), the zero-order term reads $I^0(0) = y_0 K^{\beta-1}$. Otherwise, when $\widetilde{x} \neq 0$, for $\nu = 0$ the zero-order term is $I^0(\widetilde{x}) = \frac{\widetilde{x} y_0(1-\beta)}{x_0^{1-\beta} - K^{1-\beta}}$. Otherwise, for $\nu > 0$, when $\beta < 1$, two zero-order terms exist:

- Hagan et al's:

$$I^0_H(\widetilde{x}) = \frac{\nu\widetilde{x}\zeta}{z}\left(\log\left(\frac{\sqrt{1-2\rho\zeta+\zeta^2}+\zeta-\rho}{1-\rho}\right)\right)^{-1},$$

  with

$$z = \frac{\nu}{y_0}\frac{x_0^{1-\beta} - K^{1-\beta}}{1-\beta}, \quad \text{and} \quad \zeta = \frac{\nu}{y_0}\frac{x_0 - K}{(x_0-K)^{\beta/2}};$$

- Berestycki-Busca-Florent's:

$$I^0_B(\widetilde{x}) = \nu\widetilde{x}/\log\left(\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho}\right),$$

  with z as above.

Hence, for $\beta < 1$ the two different expansions are $I_B(\widetilde{x}) = I^0_B\left(1 + I^1_H(\widetilde{x})t\right)$, and $I_H(\widetilde{x}) = I^0_H\left(1 + I^1_H(\widetilde{x})t\right)$. We implemented the BBF one one below.

For $\beta = 1$ and $\nu > 0$, the zero-order term reads

$$I^0(\widetilde{x}) = \nu\widetilde{x}/\log\left(\frac{\sqrt{1-2\rho z+z^2}+z-\rho}{1-\rho}\right),$$

with $z = \nu x_0/y_0$.

In [16]:
```
# Note that we normalise the stock price to F_0 = 1
```

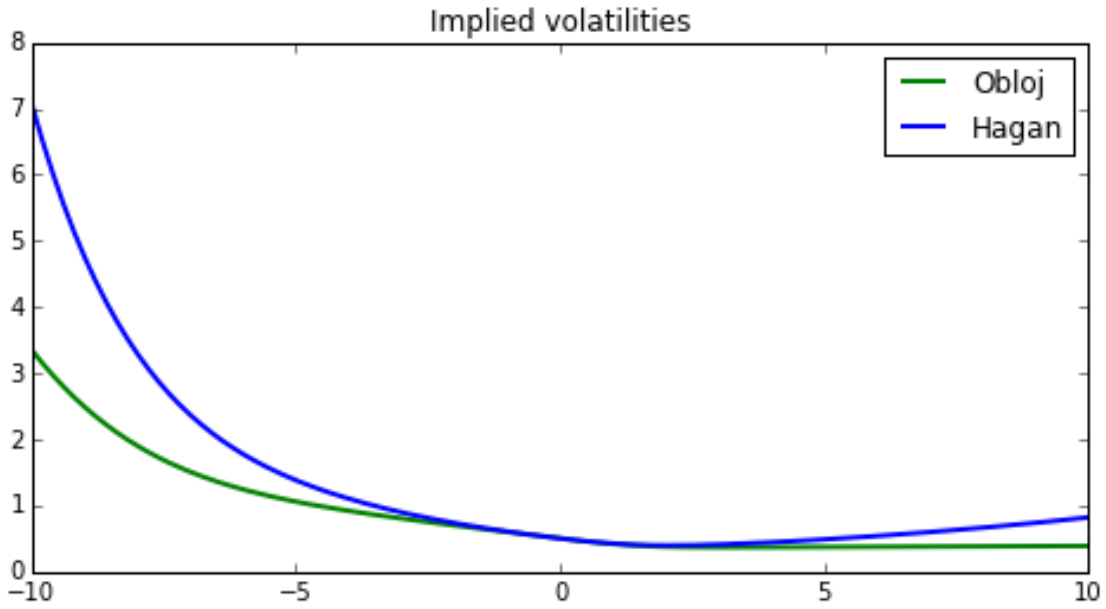
```
def sabr_vol_HKLW(y0, nu, beta, rho, x, t):
    # returns the implied volatility (square root) using Hagan et al.'s expansion
    # and the mapping x -> -x
    K = np.exp(x)
    b1 = 1. - beta
    if abs(x) < 1E-7:
        H0 = y0 * pow(K, b1)
    elif nu < 0.00000011:
```

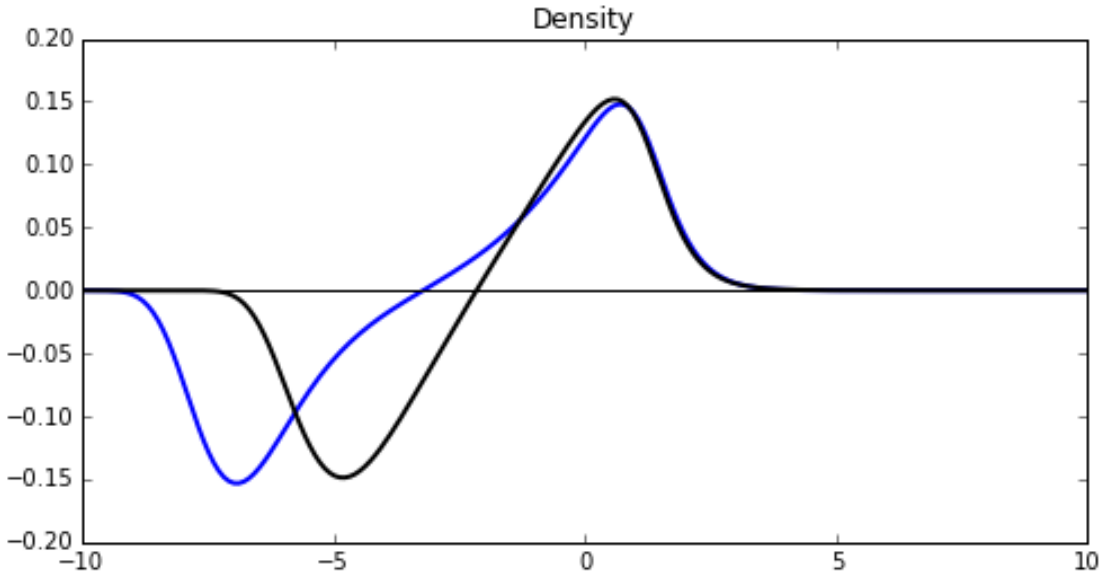
```
        H0 = -x * y0 * b1 / (1. - pow(K, b1))
    elif beta < 0.999:
        z = (nu / y0) * (1.0 - pow(K, b1)) / b1
        zeta = (nu / y0) * (1. - K) / pow(K, 0.5 * beta)
        xz = np.log(
            (np.sqrt(1. - 2. * rho * z + z * z) + z - rho) / (1. - rho))
        H0 = -(nu * x * zeta / z) / xz
    else:
        z = -nu * x / y0
        xz = np.log(
            (np.sqrt(1. - 2. * rho * z + z * z) + z - rho) / (1. - rho))
        H0 = -nu * x / xz
    H1 = (b1 * b1 / 24.) * (y0 * y0) / pow(K, b1) + (0.25 * rho * nu *
                                                     beta * y0) / (pow(K, (b1) / 2.)) + nu * nu
    return np.sqrt(H0 * (1. + H1 * t))


def sabr_vol_Obloj(y0, nu, beta, rho, x, t):
    # returns the implied volatility (square root of Obloj's formula) using Obloj's expansion
    # and the mapping x -> -x
    K = np.exp(x)
    b1 = 1. - beta
    if abs(x) < 0.000000001:
        H0 = y0 * pow(K, b1)
    elif nu < 0.00000011:
        H0 = -x * y0 * b1 / (1. - pow(K, b1))
    elif beta < 0.999:
        z = (nu / y0) * (1. - pow(K, b1)) / b1
    else:
        z = -nu / y0 * x
    xz = np.log(
        (np.sqrt(1. - 2. * rho * z + z * z) + z - rho) / (1. - rho))
    H0 = -(nu * x) / xz
    H1 = (b1 * b1 / 24.) * (y0 * y0) / pow(K, b1) + (0.25 * rho * nu *
                                                     beta * y0) / (pow(K, b1 / 2.)) + nu * nu *
    return np.sqrt(H0 * (1. + H1 * t))
```

3.1 Numerics

In [20]:
```
logStrikeMin, logStrikeMax, nbStrikes = -10., 10., 200
vMin, vMax = 1E-5, 10.
nu, beta, y0, rho, t = 0.1, 0.4, 0.25, -0.33, 15.1

logStrikes = np.linspace(
```

```
    logStrikeMin, logStrikeMax, num=nbStrikes)
oblojformula, hklwformula = [], []

for k in logStrikes:
   expansion = sabr_vol_Obloj(y0, nu, beta, rho, k, t)
   expansion2 = sabr_vol_HKLW(y0, nu, beta, rho, k, t)
   oblojformula.append(expansion)
   hklwformula.append(expansion2)

plt.figure(figsize=(8, 4))
plt.plot(
    logStrikes, oblojformula, 'g', linewidth=2, label="Obloj")
plt.plot(
    logStrikes, hklwformula, 'b', linewidth=2, label="Hagan")
plt.title('Implied volatilities')
plt.legend(loc=1)
plt.show()

eps = 1E-4
oblojSmile, oblojSmileP, oblojSmileM, haganSmile, haganSmileP, haganSmileM = [
], [], [], [], [], []

for logStrike in logStrikes:
   oblojSmile.append(
       sabr_vol_Obloj(y0, nu, beta, rho, logStrike, t))
   oblojSmileP.append(
       sabr_vol_Obloj(y0, nu, beta, rho, (logStrike + eps), t))
   oblojSmileM.append(
       sabr_vol_Obloj(y0, nu, beta, rho, (logStrike - eps), t))
   haganSmile.append(
       sabr_vol_HKLW(y0, nu, beta, rho, logStrike, t))
   haganSmileP.append(
       sabr_vol_HKLW(y0, nu, beta, rho, (logStrike + eps), t))
   haganSmileM.append(
       sabr_vol_HKLW(y0, nu, beta, rho, (logStrike - eps), t))

plt.figure(figsize=(8, 4))
plotDensity(
    oblojSmile, oblojSmileP, oblojSmileM, t, eps, logStrikes, 'b')
plotDensity(
    haganSmile, haganSmileP, haganSmileM, t, eps, logStrikes, 'k')
plt.plot(logStrikes, [0. for k in logStrikes], 'k')
plt.title('Density')
plt.show()
```


Implied volatilities
Obloj
Hagan
8
7
6
5
4
3
2
1
0
−10
−5
0
5
10


Density
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−10
−5
0
5
10

In [36]: