Fast and Accurate Analytic Basis Point Volatility

Fabien Le Floc'h

Calypso Technology, 106 rue de La Boétie, 75008 Paris

(v3.1 released June 2016)

Abstract *This paper describes a fast analytic formula to obtain the basis point volatility "b.p. vol" for a given option price under the Bachelier normal model with near machine accuracy. It handles near-the-money as well as very far out-of-the-money options and low volatilities.*

Key Words: implied volatility, b.p. vol, Bachelier

1. Introduction

With the current low interest rates, the practice of using basis point volatility "b.p. vol" has increased significantly in interest rate derivatives. The b.p. vol is simply the implied volatility of an option under the Bachelier (or normal) model.

In the Black-Scholes world, there is no simple formula to obtain the Black implied volatility out of the option price for a given strike and expiry. One has to rely on a rough approximation (Li, 2008), which can eventually be used initial guess for a solver (Li and Lee, 2011). A robust algorithm with near machine precision is given in (Jäckel, 2013).

In the Bachelier world, things are much simpler. The problem can be reduced to a single variable, allowing an analytic representation of the implied volatility (Choi *et al.*, 2009). The algorithm from Choi *et al.* (2009) works well near-the-money: it can be tuned for machine accuracy by adding one Newton step. But it can break down for low volatilities or far out-of-the-money options because of the reliance on a straddle price. Instead of relying on the straddle price, our algorithm relies directly on the call option price and uses a more precise numerical representation.

We will first present the reduction to a single variable using call options prices, explain our choice of variable transforms for in-the-money and out-of-the-money options to make the problem better behaved for a Chebyshev polynomial representation. Then we will measure the accuracy of the resulting algorithm and show a practical problem that can arise when out-of-the-money options volatilities are not computed with a high enough accuracy, e.g. with the algorithm from (Choi *et al.*, 2009). And finally we will give example code in Octave/Matlab.

The author wishes to thank Gary Kennedy for valuable feedback, especially around the representation in terms of rational functions.

Correspondence Address: Calypso Technology, 106 rue de La Boétie, 75008 Paris. Email: fabien_lefloch@calypso.com

ISSN: Print/ISSN Online/YY/issnum1stp–ppcnt© 2013 Unpublished
DOI:

2. Problem Reduction

The Bachelier model can be described by the normal process:

$$dF = \sigma dW \tag{1}$$

where W is a Brownian motion and F is typically a forward rate.

The price of put and call options of strike K and expiry T are then:

$$C = (F-K)N\left(\frac{F-K}{\sigma\sqrt{T}}\right) + \sigma\sqrt{T}n\left(\frac{F-K}{\sigma\sqrt{T}}\right)\ , \tag{2}$$

$$P = (K-F)N\left(\frac{K-F}{\sigma\sqrt{T}}\right) + \sigma\sqrt{T}n\left(\frac{K-F}{\sigma\sqrt{T}}\right)\ . \tag{3}$$

Note that under this model, the classic put-call parity relationship is preserved: $C-P = F-K$. Let's define:

$$x = F - K\ , \tag{4}$$

$$v = \sigma\sqrt{T}\ , \tag{5}$$

$$d = \frac{x}{v}\ . \tag{6}$$

We can then write the call price as

$$C(x) = xN\left(d\right) + vn(d) \tag{7}$$

and obtain the reduction

$$\frac{x}{C(x)} = \frac{d}{dN(d) + n(d)} = f(d)\ . \tag{8}$$

We now have a direct mapping between $C(x)$ and d, that is, between the call price and the volatility. Choi *et al.* (2009) present a similar formula for a straddle.

3. Variable Transforms

3.1 In-the-money

For in-the-money call options, that is when $x >= 0$, there is a natural mapping to the interval [0,1]: in-the-money call option prices verify the following inequalities:

$$0 \leq \frac{x}{C(x)} \leq 1\ . \tag{9}$$

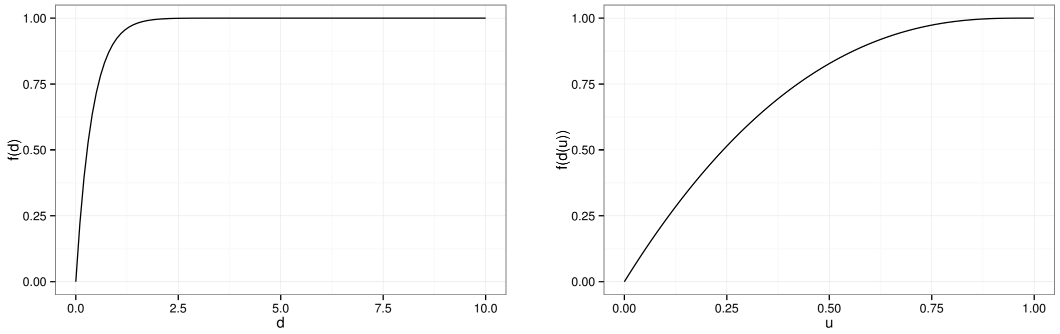
If we had infinite machine accuracy, the put-call parity relationship along with the identity $C(x) - C(-x) = x$ would reduce the problem of finding the volatility for a given option price to in-the-money call options only.

We then further proceed as Choi *et al.* (2009) by looking for a good numerical representation of h defined so that:

$$\sigma = \frac{C(x)}{\sqrt{T}}h(z) \tag{10}$$

with $z = \frac{C(x)}{x}$.

The interest of this representation is the behavior near the money: when $z \to 0$, $h(z) \to \sqrt{2\pi}$. In terms of the variable d, we have $h = \frac{f(d)}{d}$. We notice that the transform $d(u) = -log(1-u)$ makes f much more linear. $d(u)$ is not too far of f^{-1}.



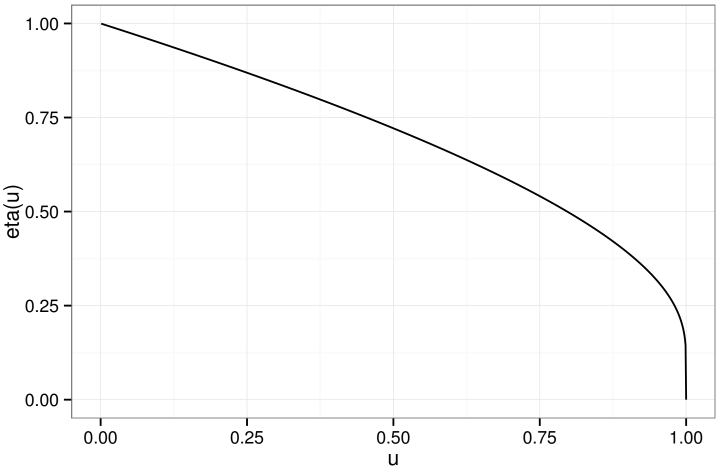
(a) $f(d)$

(b) $f(d(u))$

For a given $z = \frac{x}{C(x)}$, we have $d = f^{-1}(z)$, and then $h = \frac{z}{f^{-1}(z)}$. To make h more linear, we therefore consider the transformation

$$\eta(z) = \frac{z}{\log(1-z)} \ . \tag{11}$$

In theory η should be close to h. We then decide to compute the b.p. vol using:



$$\sigma = \frac{C(x)}{T} h\left(\eta\left(\frac{x}{C(x)}\right)\right) \ . \tag{12}$$

3.2 Out-of-the-money

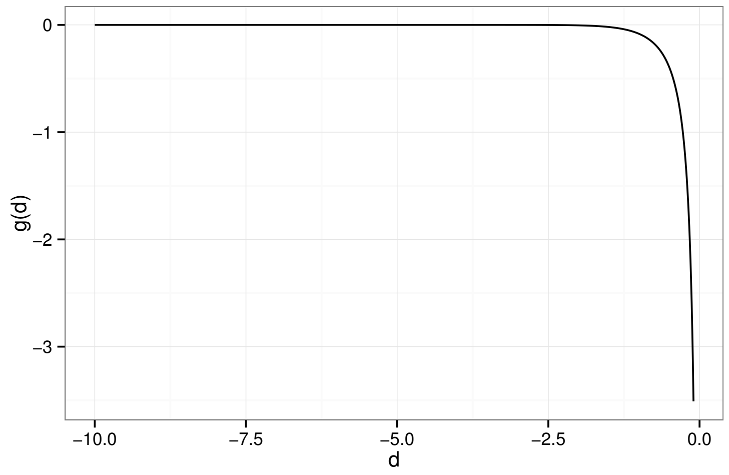
For out-of-the-money call options (when $x <= 0$) , we approximate instead directly

$$\frac{C(x)}{x} = N(d) + \frac{n(d)}{d} = g(d) \ . \tag{13}$$

We have:

$$-1 \leq \frac{C(x)}{x} \leq 0 \text{ for } d < -0.27603 \ . \tag{14}$$

Figure 1.: $g(d)$



We can switch to an in-the-money representation if $-\frac{C(x)}{x} > \alpha$ as:

$$-\frac{C(x)}{x} > \alpha \tag{15}$$

$$\iff -\frac{C(-x)}{x} - 1 > \alpha \text{ because } C(x) = C(-x) + x \tag{16}$$

$$\iff -\frac{x}{C(-x)} < \frac{1}{1+\alpha} \ . \tag{17}$$

$C(-x)$ is an in-the-money call of moneyness $-x$, and we can apply the results of the previous section for $-\frac{x}{C(-x)} \in [0, \frac{1}{1+\alpha}]$. We can therefore just focus now on approximating $-\frac{C(x)}{x}$ on $[0, \alpha]$. The cutoff α can be chosen so that the in-the-money formula is really only used near-the-money where it won't suffer from machine accuracy issues.

Figure 2 shows how the normal volatility behaves as a function of $-g$ for $g \in [-1, 0]$: its behavior near 0 is a square function in log-scale. We therefore rely on the transform

$$\tilde{\eta}(z) = -\frac{1}{\beta_e - \beta_s}(\log(z) + \beta_s) \tag{18}$$

where β_e, β_s are scaling parameters. If we choose $\beta_s = -\log(\alpha)$ and $\beta_e = 300\log(10)$, $\tilde{\eta}$ maps $[10^{-300}, \alpha]$ to $[0, 1]$. The problem is then to find an accurate numerical representation of the function:

$$\tilde{h}(u) = \left[g^{-1} \circ \tilde{\eta}^{-1}(u)\right]^2 \ . \tag{19}$$

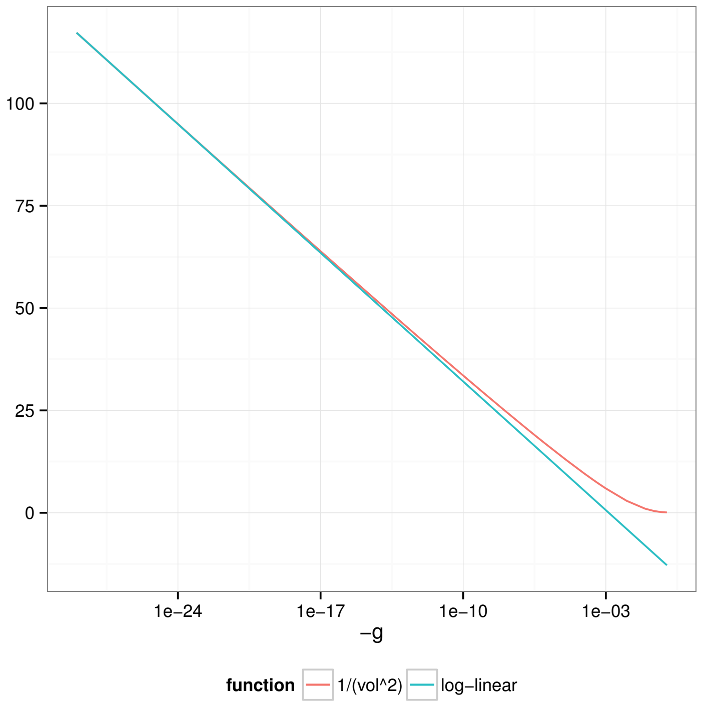
Given $\tilde{h}$, we can find d with

$$d = -\sqrt{\tilde{h} \circ \tilde{\eta}\left(\frac{-C(x)}{x}\right)} \tag{20}$$

and σ with

$$\sigma = -\frac{x}{\sqrt{\tilde{h} \circ \tilde{\eta}\left(\frac{-C(x)}{x}\right) T}} \ . \tag{21}$$

Figure 2.: volatility in terms of $-g$



4. Numerical Approximation

4.1 A table of Chebyshev polynomials

Using a single Chebyshev polynomial to represent $h(\eta)$, even of high degree, does not lead to an accurate enough approximation. Similarly to S.G. Johnson in the Faddeeva library for the computation of `erfcx` function, we can split the interval into several (in our case, four) sub-intervals and use a Chebyshev polynomial of lower degree (in our case, of degree 11) in each interval. The first of our four intervals can effectively be ignored, as the out-of-the-money formula is going to be used instead.

To calculate the coefficients of the Chebyshev polynomials, we compute $h(\eta)$ with a simple bisection method using high precision floating point arithmetic.

It turns out that, then, the accuracy can be limited by the computation of $\eta(z)$ when not using high precision arithmetic but standard IEEE 754 double-precision binary floating point format. A simple remedy is to use a Taylor expansion around 0. We have

$$\eta(z) = 1 - \frac{z}{2} - \frac{z^2}{12} - \frac{z^3}{24} - \frac{19\,z^4}{720} - \frac{3\,z^5}{160} - \frac{863\,z^6}{60480} - \frac{275\,z^7}{24192} - \frac{33953\,z^8}{3628800} - \frac{8183\,z^9}{1036800} + \cdots \quad (22)$$

The 7th order is enough to be close to double precision for $z < 10^{-2}$.

We proceed in similar fashion to approximate $\tilde{h}$, paying attention to compute successive $[\beta_{i-1}, \beta_i]$ ranges corresponding to each interval so that the error stays under 10^{-16}, this corresponds effectively to total of 24 Chebyshev polynomial of degree 11.

4.2 Rational functions

While the table of Chebyshev polynomials approach leads to a fast and accurate representation of the implied volatility accross the full range of strikes, we can improve even more the performance and the memory use with carefully chosen rational functions.

We want to approximate the target functions $h(\eta)$ and $\tilde{h}(\tilde{\eta})$, computed with a high precision floating point library, by rational functions. The Remez algorithm is a relatively popular minimax technique to find the best possible rational function approximation. Instead, like Press *et al.* (1996), we only consider the rational function solution of the weighted least squares problem on a large number of Chebyshev nodes. We found that weights of $\frac{1}{h(\eta_i)^2}$ or $\frac{1}{h(\eta_i)^4}$ helped reduce the error in terms of implied volatility.

The compromise between accuracy and number of intervals to consider led us to two different rational function representations: a first one, named "LFK-3" with a cutoff at $\alpha = 10^{-3}$ comprised of one 7/9 rational function near the money and two 9/7 rational functions out of the money and a second, more accurate representation, named "LFK-4" with a cutoff at $\alpha = 0.15$ comprised of one 7/5 rational function near the money and three 9/7 rational functions out of the money. The notation X/Y means a rational function with a numerator of degree X and denominator of degree Y.

4.2.1 LFK-3 approximation with three rational functions

The "LFK-3" approximation is:

$$\frac{\sum_{i=0}^{7} a_i \eta^i}{\sum_{i=0}^{9} b_i \eta^i} \text{ for } -\frac{C(x)}{x} \geq 10^{-3} \tag{23}$$

with

$$\begin{array}{ll}
a_0{=}0.04952850674926118, & b_0{=}1.0, \\
a_1{=}3.573874368966075, & b_1{=}6.701606616297208, \\
a_2{=}30.91406972556886, & b_2{=}336.0845085729835, \\
a_3{=}871.2008725718615, & b_3{=}86.69811786952944, \\
a_4{=}1437.861872761533, & b_4{=}21802.58386522576, \\
a_5{=}53975.49234255654, & b_5{=}51666.62041145850, \\
a_6{=}127255.5078341320, & b_6{=}20377.75740489431, \\
a_7{=}52472.70275112474, & b_7{=}\text{-}117.4240584500323, \\
 & b_8{=}9.80921194846781, \\
 & b_9{=}\text{-}0.5816685619636752,
\end{array}$$

$$\frac{\sum_{i=0}^{9} c_i \tilde{\eta}^i}{\sum_{i=0}^{7} d_i \tilde{\eta}^i} \text{ for } -\frac{C(x)}{x} < 10^{-3} \text{ and } \tilde{\eta} < 0.085 \tag{24}$$

with

$$\begin{array}{ll}
c_0{=}5.93608281722568, & d_0{=}1.0, \\
c_1{=}2634.187545515314, & d_1{=}277.0743059490384, \\
c_2{=}475307.1038126473, & d_2{=}30281.24135063437, \\
c_3{=}44058602.43544545, & d_3{=}1586389.030254879, \\
c_4{=}2176736087.988461, & d_4{=}39715052.70340591, \\
c_5{=}\ 54006392703.78609, & d_5{=}433671911.2420689, \\
c_6{=}595640924872.0572, & d_6{=}1679493127.501784, \\
c_7{=}2337443308769.071, & d_7{=}1301324393.715461, \\
c_8{=}1826625538648.927, & \\
c_9{=}\text{-}1916239413.115259, &
\end{array}$$

$$\frac{\sum_{i=0}^{9} e_i \tilde{\eta}^i}{\sum_{i=0}^{7} f_i \tilde{\eta}^i} \text{ for } -\frac{C(x)}{x} < 10^{-3} \text{ and } 0.085 \leq \tilde{\eta} \tag{25}$$

with

$$\begin{array}{ll}
e_0{=}5.935613784714764, & f_0 = 1.0 \\
e_1{=}1620.916614866792, & f_1{=}106.5670395913827, \\
e_2{=}145978.7247033772, & f_2{=}3162.004975374641, \\
e_3{=}4300446.050303745, & f_3{=}34515.54872237787, \\
e_4{=}47418292.49852913, & f_4{=}151370.8692695046, \\
e_5{=}209843464.903416, & f_5{=}266321.1526534616, \\
e_6{=}371454298.8039491, & f_6{=}170809.30608537, \\
e_7{=}239119177.6582749, & f_7{=}29414.21916215061, \\
e_8{=}41259962.7354683, & \\
e_9{=}244.7680086009763. &
\end{array}$$

The above equations are used for $x \leq 0$. We make use of the identity $C(x) - C(-x) = x$ to compute the price for $x > 0$.

4.2.2 LFK-4 representation with four rational functions

The "LFK-4" representation is:

$$\frac{\sum_{i=0}^{7} a_i \eta^i}{\sum_{i=0}^{5} b_i \eta^i} \text{ for } -\frac{C(x)}{x} \geq 0.15 \tag{26}$$

with

$$\begin{array}{ll}
a_0{=}0.06155371425063157, & b_0{=}1.0, \\
a_1{=}2.723711658728403, & b_1{=}1.436062756519326, \\
a_2{=}10.83806891491789, & b_2{=}118.6674859663193, \\
a_3{=}301.0827907126612, & b_3{=}441.1914221318738, \\
a_4{=}1082.864564205999, & b_4{=}313.4771127147156, \\
a_5{=}790.7079667603721, & b_5{=}40.90187645954703, \\
a_6{=}109.330638190985, & \\
a_7{=}0.1515726686825187, &
\end{array}$$

$$\frac{\sum_{i=0}^{9} c_i \tilde{\eta}^i}{\sum_{i=0}^{7} d_i \tilde{\eta}^i} \text{ for } -\frac{C(x)}{x} < 0.15 \text{ and } \tilde{\eta} < 0.01 \tag{27}$$

with

$$\begin{array}{ll}
c_0{=}0.6409168551974356, & d_0{=}1.0, \\
c_1{=}776.7622553541449, & d_1{=}640.570978803313, \\
c_2{=}431496.7672664836, & d_2{=}206873.1616020722, \\
c_3{=}142810081.2530825, & d_3{=}40411807.74439474, \\
c_4{=}30593703611.75923, & d_4{=}5007804896.265911, \\
c_5{=}4296256150040.825, & d_5{=}379858284395.2218, \\
c_6{=}377808909050483.9, & d_6{=}15253797078346.91, \\
c_7{=}1.799539603508817\text{e}{+}16, & d_7{=}211469320780659.9, \\
c_8{=}2.864267851212242\text{e}{+}17, & \\
c_9{=}1.505975341130321\text{e}{+}16, &
\end{array}$$

$$\frac{\sum_{i=0}^{9} e_i \tilde{\eta}^i}{\sum_{i=0}^{7} f_i \tilde{\eta}^i} \text{ for } -\frac{C(x)}{x} < 0.15 \text{ and } 0.01 \leq \tilde{\eta} < 0.09 \tag{28}$$

with

$$
\begin{array}{ll}
e_0{=}0.6421698396894946, & f_0 = 1.0 \\
e_1{=}639.0799338046976, & f_1{=}428.4860093838116, \\
e_2{=}278070.4504753253, & f_2{=}86806.89002606465, \\
e_3{=}64309618.34521588, & f_3{=}8635134.393384729, \\
e_4{=}8434470508.516712, & f_4{=}368872214.1525768, \\
e_5{=}429163238246.6056, & f_5{=}6359299149.626331, \\
e_6{=}8127970878235.127, & f_6{=}39926015967.88848, \\
e_7{=}53601225394979.81, & f_7{=}67434966969.06365, \\
e_8{=}92738918006503.35, & \\
e_9{=}54928597545.97237, &
\end{array}
$$

$$
\frac{\sum_{i=0}^{9} g_i \tilde{\eta}^i}{\sum_{i=0}^{7} h_i \tilde{\eta}^i} \text{ for } -\frac{C(x)}{x} < 0.15 \text{ and } 0.09 \leq \tilde{\eta} \tag{29}
$$

with

$$
\begin{array}{ll}
g_0{=}0.9419766804760195, & h_0 = 1.0 \\
g_1{=}319.5904313022832, & h_1{=}170.3825619167351, \\
g_2{=}169280.1584005307, & h_2{=}6344.159541465554, \\
g_3{=}7680298.116948191, & h_3{=}78484.04408022196, \\
g_4{=}102052455.1237945, & h_4{=}370696.1131305614, \\
g_5{=}497528976.6077898, & h_5{=}682908.5433659635, \\
g_6{=}930641173.0039455, & h_6{=}451067.0450625782, \\
g_7{=}619268950.1849232, & h_7{=}79179.06152239779, \\
g_8{=}109068992.0230439, & \\
g_9{=}672.856898188759. &
\end{array}
$$

The above equations are used for $x \leq 0$. We make use of the identity $C(x) - C(-x) = x$ to compute the price for $x > 0$.

4.3 Accuracy

When the algorithm from Choi *et al.* (2009) is applied out-of-the-money call options, internally in-the-money put options are used to compute the equivalent straddle price through the put-call parity relationship. Unfortunately, because numbers are represented with limited accuracy in the 64-bit double precision standard (near 1.0, the maximum accuracy is machine epsilon, that is around 10^{-16}, while near 0 it is 10^{-308}), this makes the algorithm fail for very far out-of-the-money options (see Table 1) as soon as the price is under the unit of least precision at x.

The Chebyshev and LFK-4 normal volatility methods are accurate, near the money as well as very far out of the money with an error in volatility of less than 10^{-15} (Figures 3 and 4). The LFK-3 approximation is accurate up to $3 \cdot 10^{-14}$, also accross the full range of strikes. In our setting, that is $\sigma = 1.0, T = 1.0, F = 1.0$, a call option of strike $K = 38$ has a value of less than 10^{-300}.

One can wonder if being accurate so far away is of any practical use. Unfortunately, it is. As an illustration, we take the example of calibrating the arbitrage-free SABR model (Hagan *et al.*, 2014) to market normal volatility for 1m5y swaptions on the 28 May 2014.

Figure 5 shows that in the zone of the first strike, the CKK formula was not able to compute the normal volatility. As the calibration process is to minimize the mean square error between the model normal volatilities and the market normal volatilities, the error measure will be wrong, and the wrong parameters will be selected. This is because, while the market price of a swaption at the first strike is not so low, the calibrated swaption price is extremely low

Table 1.: Error in terms of volatility using a reference $\sigma = 1.0$, $T = 1.0$, $F = 1.0$. CKK denotes the algorithm from (Choi *et al.*, 2009), CKK-Newton does one extra Newton iteration, and Chebyshev is our approximation.

Strike	CKK	CKK-Newton	Chebyshev	LFK-3	LFK-4
1.000010	1.11e-15	0.00e+00	3.33e-16	9.33e-15	4.44e-16
1.006660	8.88e-16	0.00e+00	2.22e-16	2.22e-15	0.00e+00
2.000000	3.03e-13	2.22e-16	2.22e-16	1.51e-14	2.22e-16
4.000000	3.82e-11	3.15e-14	2.22e-16	1.52e-14	1.11e-16
8.800000	1.86e-03	1.86e-03	2.22e-16	3.33e-15	1.11e-16
9.000000	1.00e+00	NaN	2.22e-16	9.33e-15	2.22e-16
30.000000	1.00e+00	NaN	0.00e+00	2.22e-16	2.22e-16

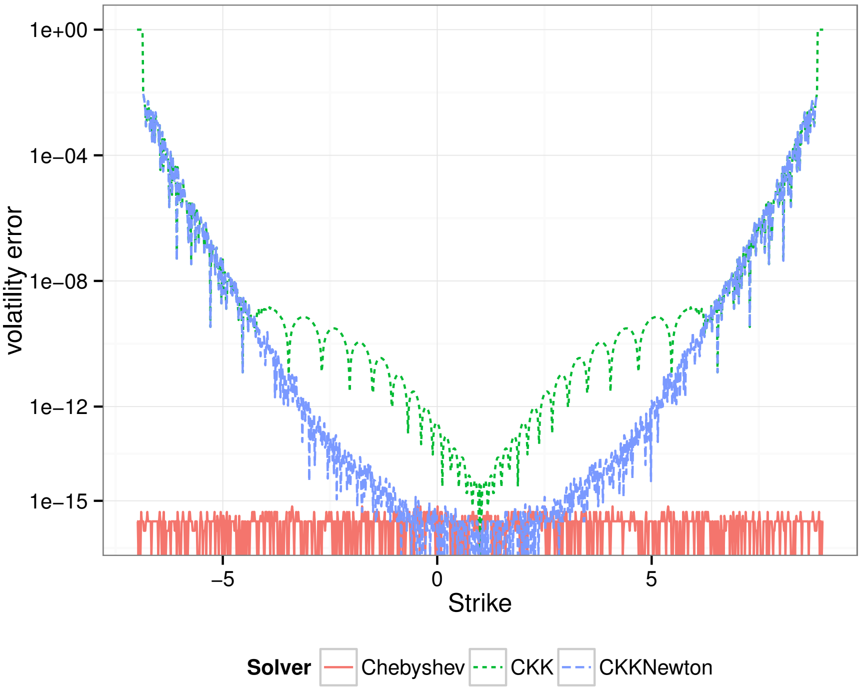


Figure 3.: Error against strike in double precision arithmetic.

(under machine epsilon), in a zone where the CKK algorithm fails. Our Chebyshev based algorithm allow proper calibration, while keeping roughly the same performance (Table 2). LFK-3 and LFK-4 are around 30% faster on a Intel Core i5-3210M 2.50GHz CPU with the Go 1.6 language. It can be interesting to compare the time taken to imply the volatility with LFK-4 (the inverse function) compared to the time taken to compute the option price with the optimized Bachelier implementation described in appendix A (the direct function): the inverse function takes 0.11 microsecond while the direct function takes 0.10 microsecond.

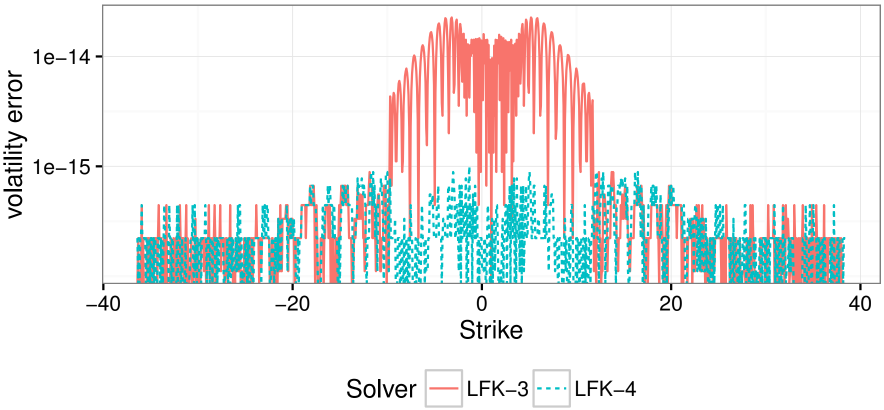
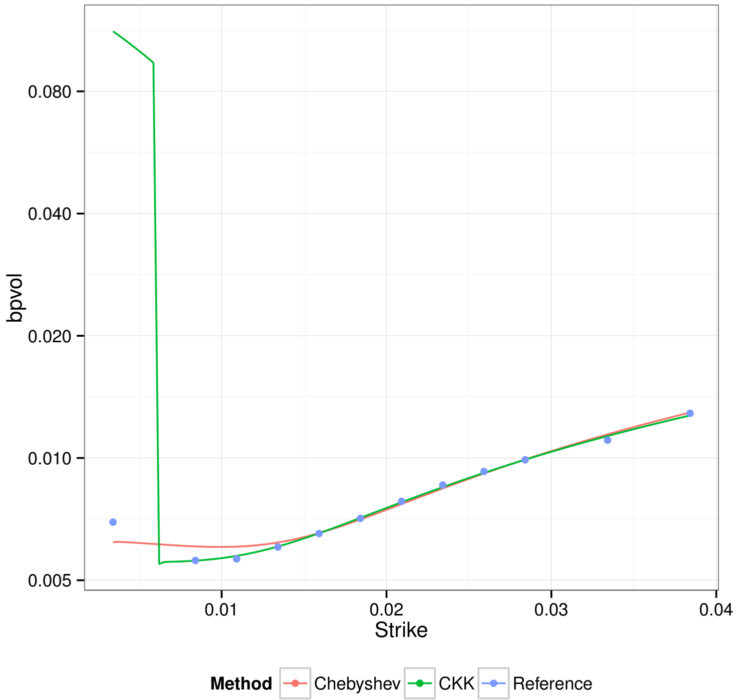


Figure 4.: Error of the rational approximations against a very wide range of strikes.

Figure 5.: bpvol resulting from a calibrated SABR with different normal volatility algorithms



5. Conclusion

The Bachelier volatility can be implied from option prices with close to machine epsilon accuracy accross the full range of strikes with a formula consisting in four rational functions. This allows the calibration of volatility models to very out-of-the-money strikes. With the proposed formula, the Bachelier implied volatility function can be considered as essentially another special function.

Table 2.: Time taken to imply the volatility of 1 million options within 3 standard deviations using a reference $\sigma = 1.0$, $T = 1.0$, $F = 1.0$ with double precision arithmetic. CKK denotes the algorithm from (Choi *et al.*, 2009), CKK-Newton does one extra Newton iteration, Chebyshev is the Chebyshev table approximation, LFK-3 and LFK-4 are the two rational function approximations.

Method	Vol RMSE	Time (s)
CKK	6e-11	0.14
CKK-Newton	9e-15	0.25
Chebyshev	5e-16	0.14
LFK-3	2e-14	0.11
LFK-4	7e-16	0.11

Appendix A. Fast and Accurate computation of the option price under the Bachelier model

Jäckel (2013) presents a small trick to compute more efficiently an option price under Black-Scholes. The idea is to use directly the scaled complementary error function `erfcx` instead of the cumulative normal distribution function. In several algorithms to compute the cumulative normal distribution, `erfcx` is computed behind the scenes. This is notably the case in the Faddeeva package (Johnson, 2014), an algorithm that improves in terms of performance over the popular Cody algorithm from SPECFUN Fortran library (Cody, 1993) (where `erfcx` is computed for $x > 0.46875$), as well as in (Schonfelder, 1978; Hill, 1973). As a result, the exponential scaling can be factored between the two cumulative normal distribution evaluations as we have:

$$N(x) = \frac{1}{2} e^{-\frac{1}{2}x^2} \operatorname{erfcx}(-\frac{x}{\sqrt{2}}) \ . \tag{A1}$$

The same strategy is also applicable to the Bachelier formula, factoring the exponential between the cumulative normal distribution and the normal density calls:

$$C(x) = n(d) \left(x\sqrt{\frac{\pi}{2}} \operatorname{erfcx}(-\frac{d}{\sqrt{2}}) + v \right) \ . \tag{A2}$$

In practice, the overhead of one call to `exp` is significant in the evaluation of the Bachelier or Black-Scholes formulas.

Appendix B. Example code

We present here code to compute the implied volatility from a given Bachelier price as well as to compute a Bachelier price using only one call to `exp` in Matlab/Octave. The code is not optimized, but very little additional work should be needed to port it to another language in an optimized manner. Our performance results come from optimized Java code.

Listing 1: volBachelierLFK4.m - Matlab/Octave code computing the implied volatility for a given option price

```
function vol = volBachelierLFK4(sign, strike, forward, tte, price)
  aLFK4 = [0.06155371425063157, 2.723711658728403, 10.83806891491789, 301.0827907126612, 1082.864564205999,
        790.7079667603721, 109.330638190985, 0.1515726686825187, 1.436062756519326, 118.6674859663193,
        441.1914221318738, 313.4771127147156, 40.90187645954703];
  bLFK4 = [0.6409168551974357, 788.5769356915809, 445231.8217873989, 149904950.4316367, 32696572166.83277,
        4679633190389.852, 420159669603232.9, 2.053009222143781e+16, 3.434507977627372e+17, 2.012931197707014e
        +16, 644.3895239520736, 211503.4461395385, 42017301.42101825, 5311468782.258145, 411727826816.0715,
        17013504968737.03, 247411313213747.3];
  cLFK4 = [0.6421106629595358, 654.5620600001645, 291531.4455893533, 69009535.38571493, 9248876215.120627,
       479057753706.175, 9209341680288.471, 61502442378981.76, 107544991866857.5, 63146430757.94501,
       437.9924136164148, 90735.89146171122, 9217405.224889684, 400973228.1961834, 7020390994.356452,
       44654661587.93606, 76248508709.85633];
  dLFK4 = [0.936024443848096, 328.5399326371301, 177612.3643595535, 8192571.038267588, 110475347.0617102,
       545792367.0681282, 1033254933.287134, 695066365.5403566, 123629089.1036043, 756.3653755877336,
       173.9755977685531, 6591.71234898389, 82796.56941455391, 396398.9698566103, 739196.7396982114,
       493626.035952601, 87510.31231623856];
  betaStart = -log(0.15); betaEnd = -log(realmin());
  x = (forward-strike)*sign;
  if (abs(x) < eps)
    vol = price*sqrt(2*pi/tte);
    return;
  endif
  if (x > 0)
    z = (price-x)/x;
  else
    z = -price/x;
  endif
  if (z < 0.15)
    u = -(log(z)+betaStart)/(betaEnd-betaStart);
    if (u < 0.0091)
      num = bLFK4(1) + u*(bLFK4(2)+u*(bLFK4(3)+u*(bLFK4(4)+u*(bLFK4(5)+u*(bLFK4(6)+u*(bLFK4(7)+u*(bLFK4(8)+u*(
           bLFK4(9)+u*bLFK4(10)))))))));
      den = 1.0 + u*(bLFK4(11)+u*(bLFK4(12)+u*(bLFK4(13)+u*(bLFK4(14)+u*(bLFK4(15)+u*(bLFK4(16)+u*(bLFK4(17))))
           ))));
      hz = num / den;
    elseif (u < 0.088)
      num = cLFK4(1) + u*(cLFK4(2)+u*(cLFK4(3)+u*(cLFK4(4)+u*(cLFK4(5)+u*(cLFK4(6)+u*(cLFK4(7)+u*(cLFK4(8)+u*(
           cLFK4(9)+u*cLFK4(10)))))))));
      den = 1.0 + u*(cLFK4(11)+u*(cLFK4(12)+u*(cLFK4(13)+u*(cLFK4(14)+u*(cLFK4(15)+u*(cLFK4(16)+u*(cLFK4(17))))
           ))));
      hz = num / den;
    else
      num = dLFK4(1) + u*(dLFK4(2)+u*(dLFK4(3)+u*(dLFK4(4)+u*(dLFK4(5)+u*(dLFK4(6)+u*(dLFK4(7)+u*(dLFK4(8)+u*(
           dLFK4(9)+u*dLFK4(10)))))))));
      den = 1.0 + u*(dLFK4(11)+u*(dLFK4(12)+u*(dLFK4(13)+u*(dLFK4(14)+u*(dLFK4(15)+u*(dLFK4(16)+u*(dLFK4(17))))
           ))));
      hz = num / den;
    endif
    vol = abs(x) / (sqrt(hz * tte));
  else
    if (x < 0)
      price = price - x;
    endif
    z = abs(x) / price;
    u = eta(z);
    num = aLFK4(1) + u*(aLFK4(2)+u*(aLFK4(3)+u*(aLFK4(4)+u*(aLFK4(5)+u*(aLFK4(6)+u*(aLFK4(7)+u*(aLFK4(8)))))))
         ;
    den = 1.0 + u*(aLFK4(9)+u*(aLFK4(10)+u*(aLFK4(11)+u*(aLFK4(12)+u*(aLFK4(13))))));
    vol = price * num / den / sqrt(tte);
  endif
end
function eta = eta(z)
  if (z < 1e-2)
    eta = 1-z*(0.5+z*(1.0/12+z*(1.0/24+z*(19.0/720+z*(3.0/160+z*(863.0/60480+z*(275.0/24192)))))));
  else
    eta = -z/log1p(-z);
  endif
end
```

Listing 2: priceBachelier.m - price an option under the Bachelier model

```
function price = priceBachelier(sign, strike, forward, tte, vol)
  v = vol * sqrt(tte);
  x = sign * (forward-strike);
  if (abs(x) < eps)
    price = v/sqrt(2*pi);
    return;
  endif
  parity = 0;
  if (x > 0)
    parity = x;
    x = -x;
  endif
  d = x / v;
  price = parity +  normpdf(d) * (x*0.5*sqrt(2*pi)*erfcx(-d/sqrt(2)) + v);
end
```

Listing 3: volBachelierChebyshev.m - - Matlab/Octave code computing the implied volatility for a given option price through Chebyshev polynomials

```
function vol = volBachelierChebyshev(sign, strike, forward, tte, price)
 xs = [0, 0.4217386245727539, 0.55084228515625, 0.7421875, 1.0];
  Ci = [1.1457649438056933e+00, 4.9389493395341327e-01, 9.5084828942661077e-03, -5.0635553366237132e-03,
       4.7842388345056196e-03, -4.7776836566224226e-03, 4.3114855750923551e-03, -3.4655148871632008e-03,
       2.5025695403863585e-03, -1.6283830151811076e-03, 9.3498608459278867e-04, -4.1600707530037826e-04;
 2.46312728450416518e+00, 1.58182641253635958e-01, 1.78609193027728117e-04, -5.24205756984930796e-06,
      2.42783643139975411e-07, -1.26411952587526635e-08, 6.62072274170638910e-10, -3.28548007462259518e-11,
      1.43435488794632436e-12, -4.43939091051224089e-14, -4.66788255924194092e-16, 2.51814352897481322e-16;
3.25251121005064947e+00, 2.36620165648468411e-01, 2.76097303278112960e-04, -7.93794944867281994e-06,
     3.82951035161601677e-07, -2.23413365448680311e-08, 1.39731324873485435e-09, -8.89821509967213342e-11,
     5.61580195125996316e-12, -3.44275787487274692e-13, 2.00275088405750998e-14, -1.06027978363402757e-15;
4.36904846957001158e+00, 3.21751325652842701e-01, 3.61642571169418421e-04, -9.31203489051275572e-06,
     4.04566190562942834e-07, -2.21891236355190701e-08, 1.36269939039333248e-09, -8.84037949667989474e-11,
     5.87957127773745941e-12, -3.94308493169714526e-13, 2.63924286682272227e-14, -1.74264358622528222e-15];
  multiplier = [0, 0.05263201943695479, 0.21052807774781915, 0.5614082073275177, 0.816593756112753,
       1.1877727361640043, 1.6525533720542671, 2.1152683162294617, 2.7075434447737114, 3.5363832748064796,
       4.618949583420708, 6.03291374161072, 7.960958339445074, 10.505182138649168, 13.791418602534293,
       18.10565724230143, 23.769478225790593, 31.61031909507736, 42.25713675373266, 57.16156940002939,
       77.94067518725711, 107.4899008643115, 149.65283274650574, 210.7177734375, 300];
  beta = multiplier(end:-1:1) .* log(10);
  embeta = exp(-beta);
  Co = [2.30600323192718861e+03, 2.05043083247075430e+02, 2.39536277360759692e-02, -1.42320183116055924e-03,
       9.50461092350134341e-05, -6.76621453751332676e-06, 5.01484602302735643e-07, -3.82131995214250120e-08,
       2.97139625170794220e-09, -2.34640960131565624e-10, 1.87540752543154288e-11, -1.50373459515431430e-12;
1.61575635045128161e+03, 1.40084371997021321e+02, 2.26962330718600939e-02, -1.31117114957347044e-03,
     8.51093520283368612e-05, -5.88732846343277027e-06, 4.23899643148261688e-07, -3.13740847135900273e-08,
     2.36919630414626526e-09, -1.81662888200026136e-10, 1.40970338433297985e-11, -1.09771158678507572e-12;
1.14244577839507951e+03, 9.65746640508781269e+01, 2.14832644611544811e-02, -1.20564164636571738e-03,
     7.59850454702649215e-05, -5.10147035894696720e-06, 3.56396444528921101e-07, -2.55872295482973635e-08,
     1.87386926488722860e-09, -1.39317682354677504e-10, 1.04808434868854032e-11, -7.91370824068396682e-13;
8.14221183081155800e+02, 6.75408915344469563e+01, 2.05749962283125476e-02, -1.12765405950729131e-03,
     6.93590677615677035e-05, -4.54215064646345091e-06, 3.09391570448861576e-07, -2.16498374498513000e-08,
     1.54488140956553519e-09, -1.11884629996816779e-10, 8.19721227056071115e-12, -6.02828138980731756e-13;
5.84426844422268118e+02, 4.73595395993986656e+01, 1.94939514080227326e-02, -1.03704438926066275e-03,
     6.18556573238324462e-05, -3.92541502257179573e-06, 2.58960138274379020e-07, -1.75416842073835410e-08,
     1.21122172934082963e-09, -8.48498666448666160e-11, 6.01114446148331094e-12, -4.27486159462142734e-13;
4.22034946751639438e+02, 3.38390285345843012e+01, 1.89105365344666258e-02, -9.87111358507962184e-04,
     5.76987137039118155e-05, -3.58489169958060618e-06, 2.31361798193637568e-07, -1.53219693536025759e-08,
     1.03372328551015628e-09, -7.07210809686948194e-11, 4.89072696563857726e-12, -3.39455433855193462e-13;
3.06268330750858979e+02, 2.40471850984675086e+01, 1.79105848598168256e-02, -9.05304056844595767e-04,
     5.11548195499361534e-05, -3.06854551496427466e-06, 1.90998844104128161e-07, -1.21884793208863571e-08,
     7.91762537850722766e-10, -5.21181606196934139e-11, 3.46564490970099708e-12, -2.31243049172151419e-13;
2.23005332604247087e+02, 1.75862405473772547e+01, 1.77868230923009969e-02, -8.89981137147263465e-04,
     4.96701566718315586e-05, -2.93775321393517272e-06, 1.80040252307262132e-07, -1.12982470917758534e-08,
     7.20946540939965004e-10, -4.65702138075843620e-11, 3.03604866667469958e-12, -1.98471412648994867e-13;
1.62667972355314561e+02, 1.25851758168342602e+01, 1.67593410462559926e-02, -8.06865964259929533e-04,
     4.31997788710271265e-05, -2.44544045342883331e-06, 1.43159464693583910e-07, -8.56690020068779026e-09,
     5.20470929477186976e-10, -3.19626959072382770e-11, 1.97822139293063986e-12, -1.22657622803902416e-13;
1.18566493530120553e+02, 9.46659775809801829e+00, 1.73770992840480074e-02, -8.42034820952365952e-04,
     4.51933336800977188e-05, -2.55638600031097696e-06, 1.49131751734994084e-07, -8.87090260969446832e-09,
     5.34450977458847880e-10, -3.24729666711432765e-11, 1.98391152291943098e-12, -1.21158736934964368e-13;
8.54623767868520474e+01, 7.08648876487567403e+00, 1.80734494773889948e-02, -8.79158514673574593e-04,
     4.71032350485270146e-05, -2.64751380023519790e-06, 1.52833265837117697e-07, -8.96059077944794021e-09,
     5.30015695740678664e-10, -3.14882916760790340e-11, 1.87290244808356351e-12, -1.10860466510251981e-13;
6.05832566290490249e+01, 5.35345537208291944e+00, 1.95356468722009541e-02, -9.66658570039372056e-04,
     5.22578043412872984e-05, -2.94286978307513055e-06, 1.69063278706094856e-07, -9.79612142108334273e-09,
     5.68362021009136972e-10, -3.28389051927897973e-11, 1.88036701042893552e-12, -1.05856918363037906e-13;
4.20438717065676855e+01, 3.91746142694269928e+00, 2.02750971539916586e-02, -9.91367660155324879e-04,
     5.23131418135476319e-05, -2.84242481658860988e-06, 1.55624212074750491e-07, -8.47220451755257609e-09,
     4.53656931664751582e-10, -2.36134334860705219e-11, 1.17534045690193341e-12, -5.43001544535916602e-14;
2.87418835904141439e+01, 2.73641102606634634e+00, 1.92845978996126176e-02, -8.81921217829938252e-04,
     4.27034371821736626e-05, -2.08572918672709561e-06, 1.00044714573384490e-07, -4.59988790004481300e-09,
     1.95806940270498618e-10, -7.16309558001473350e-12, 1.71258929777425809e-13, 3.83833731077658905e-15;
1.93543305481815686e+01, 1.95906208202102228e+00, 1.93135538600447271e-02, -8.34150301839725063e-04,
     3.69447482819965027e-05, -1.58223352984381638e-06, 6.20536482013937047e-08, -1.99984938451233229e-09,
     3.16007760447780798e-11, 2.42628930751835339e-12, -3.40720384263683124e-13, 2.79759254187625859e-14;
1.27082048601427715e+01, 1.36630289513067926e+00, 1.85821399756413361e-02, -7.24350366367430766e-04,
     2.72372921197427225e-05, -8.83395001301227846e-07, 1.82450391048389667e-08, 4.19109646214676921e-10,
     -8.31011627960832627e-11, 6.63926417834988853e-12, -4.03348029883486892e-13, 2.01080319521359718e-14;
8.25768664193382484e+00, 8.64284873544575172e-01, 1.44356983322932973e-02, -4.40953083333075742e-04,
     1.13797502401283676e-05, -1.48931535904666241e-07, -7.64385198930138952e-09, 8.14837492189014312e-10,
     -4.64009806227305032e-11, 1.91657811562235157e-12, -5.19516934912295108e-14, -7.71832950452140827e-17;
5.37399664023837076e+00, 5.80508797906514662e-01, 1.22255955381050784e-02, -2.90168333061641029e-04,
     3.97210055405662473e-06, 1.13122593045570815e-07, -1.19881618657853447e-08, 5.62804550979071614e-10,
     -1.59217445115466664e-11, 4.41188523272878353e-14, 2.71523518301310337e-14, -1.98377652640796949e-15;
3.26687382369443746e+00, 4.69794428067438763e-01, 1.60977402884881701e-02, -3.17785406710487833e-04,
     -2.78170698806923216e-06, 6.51860031800023276e-07, -3.31395243703917697e-08, 6.51855655888523646e-10,
     3.71668186551210071e-11, -4.28550383884439312e-12, 2.07377016077089813e-13, -3.30082019935399927e-15;
1.78792847895452378e+00, 2.74109133572780550e-01, 1.20722645379343248e-02, -8.29580374604372054e-05,
     -8.56476604286481894e-06, 4.31400725863084110e-07, -3.36823221341919302e-09, -7.00281644535294660e-10,
     4.22311998791719254e-11, -6.13800490236356025e-13, -6.65355859554031427e-14, 5.16074398092519539e-15;
9.89202680545164026e-01, 1.31616375453412121e-01, 5.76124504188479415e-03, 2.57095969945979021e-05,
     -3.38132571124674654e-06, 3.80592728338321150e-08, 2.70328591323312130e-09, -1.13648818350287395e-10,
     -2.98043081534560856e-13, 1.60271224509135513e-13, -4.43832503593936826e-15, -8.94170523415808615e-17;
4.97231967506140016e-01, 1.10588522091192326e-01, 9.28845342365719299e-03, 2.35157347934832338e-04,
     -1.06786619980406935e-05, -3.85205611446075403e-07, 3.05681991112453435e-08, 4.74449512974709131e-10,
     -9.64080655394661096e-11, 7.96440770014365182e-13, 2.76386293521114245e-13, -9.64490142265403726e-15;
2.34773761638268075e-01, 2.82691112989171545e-02, 1.38574497701665850e-03, 2.88585623712612742e-05,
     -8.80883333178174867e-08, -1.33101610268625247e-08, 1.12451491663278291e-11, 7.87035584116022092e-12,
     -1.13117181931959606e-14, -5.15279626949370386e-15, 1.86890463893239915e-17, 3.52592989927947130e-18;
1.66410700339358342e-01, 7.13999302958060519e-03, 1.28165917685898399e-04, 1.07103294197533383e-06,
     1.06129363753704020e-09, -5.09469300094657841e-11, -1.78866252343168219e-13, 3.13031595844065657e-15,
     1.64297388388154339e-17, -2.18231596497765135e-19, -1.39959311193082899e-21, 1.63984108260708571e-23];
  x = (forward-strike)*sign;
  if (abs(x) < eps)
    vol = price*sqrt(2*pi/tte);
    return;
  endif
  if (x > 0)
```

```
    z = (price-x)/x;
  else
    z = -price/x;
  endif
  if (z < 0.2) %out of the money
    j = lookup(embeta, z);
    u = -(log(z)+beta(j+1))/(beta(j)-beta(j+1));
    coeff = Co(j,:);
    hz = chebyshev(u, coeff, 0, 1);
    vol = abs(x) / sqrt(hz*tte);
  else %in the money
    if (x < 0)
      price = price - x;
    endif
    z = abs(x)/price;
    u = eta(z);
    index = lookup(xs, u);
    coeff = Ci(index,:); % matrix indices start at 1
    hz = chebyshev(u, coeff, xs(index), xs(index+1));
    vol = price * hz / sqrt(tte);
  endif
end
function eta = eta(z)
  if (z < 5e-2)
    eta = 1-z*(0.5+z*(1.0/12+z*(1.0/24+z*(19.0/720+z*(3.0/160+z*(863.0/60480+z*(275.0/24192+z*(33953.0/3628800+
         z*(8183.0/1036800)))))))));
  else
    eta = -z/log1p(-z);
  endif
end
function value = chebyshev(x, c, a, b)
  y = (2*x-a-b)/(b-a);
  y2 = 2*y;
  b0 = c(12);
  b1 = 0.0;
  b2 = 0.0;
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(11);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(10);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(9);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(8);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(7);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(6);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(5);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(4);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(3);
  b2 = b1; b1 = b0; b0 = y2 * b1 - b2 + c(2);
  value = y*b0 - b1 +0.5*c(1);
end
```

References

Choi, J., Kim, K. and Kwak, M. (2009) Numerical approximation of the implied volatility under arithmetic Brownian motion, *Applied Mathematical Finance*, 16(3), pp. 261–268.

Cody, W. J. (1993) Algorithm 715: SPECFUN–a portable FORTRAN package of special function routines and test drivers, *ACM Transactions on Mathematical Software (TOMS)*, 19(1), pp. 22–30.

Hagan, P. S., Kumar, D., Lesniewski, A. S. and Woodward, D. E. (2014) Arbitrage free SABR, *Wilmott magazine*.

Hill, I. D. (1973) Algorithm AS 66: The normal integral, *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 22(3), pp. 424–427.

Jäckel, P. (2013) Lets be rational. , `http://www.pjaeckel.webspace.virginmedia.com/LetsBeRational.pdf` (accessed 03-April-2014).

Johnson, S. G. (2014) Faddeeva package. , `http://ab-initio.mit.edu/wiki/index.php/Faddeeva_Package` (accessed 03-April-2014).

Li, M. (2008) Approximate inversion of the Black–Scholes formula using rational functions, *European Journal of Operational Research*, 185(2), pp. 743–759.

Li, M. and Lee, K. (2011) An adaptive successive over-relaxation method for computing the Black–Scholes implied volatility, *Quantitative Finance*, 11(8), pp. 1245–1269.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1996) *Numerical recipes in C*, Vol. 2 (Citeseer).

Schonfelder, J. (1978) Chebyshev expansions for the error and related functions, *Math. Comp*, 32(144), pp. 1232–1240.