October 15, 2019

A PDE METHOD FOR ESTIMATION OF IMPLIED VOLATILITY

IVAN MATIĆ, RADOŠ RADOIČIĆ, AND DAN STEFANICA

Abstract. In this paper it is proved that the Black-Scholes implied volatility satisfies a second order non-linear partial differential equation. The obtained PDE is then used to construct an algorithm for fast and accurate polynomial approximation for Black-Scholes implied volatility that improves on the existing numerical schemes from literature, both in speed and parallelizability. We also show that the method is applicable to other problems, such as approximation of implied Bachelier volatility.

1. Introduction and formulation of the problem

The price of an asset in the Black-Scholes model [4] is governed by several components. Volatility is the parameter that controls the random component of the dynamics. All other parameters can be obtained from historical data by analysis of the risk-free interest rate, expected returns, and dividend payments. The market prices of derived securities such as call and put options uniquely[1] determine the value for volatility for which the Black-Scholes equation is satisfied. This unique value is called *the Black-Scholes market implied volatility*, or *implied volatility* for short.

After a standard dimension reduction, fully described in Section 4, implied volatility is equal to $\Sigma_{BS}(k,c)/\sqrt{T}$, where the integrated volatility $\Sigma = \Sigma_{BS}(k,c)$ is a function of log-moneyness k and normalized price c of the underlying asset.[2] We will prove that Σ solves the partial differential equations

$$\Sigma^3\Sigma_{cc} = \Sigma_c^2\left(\frac{\Sigma^4}{4} - k^2\right), \tag{1}$$

$$\Sigma^3\Sigma_{kk} = \left(\frac{\Sigma^2\Sigma_k}{2} + \Sigma - k\Sigma_k\right)\left(\frac{\Sigma^2\Sigma_k}{2} - \Sigma + k\Sigma_k\right), \quad \text{and} \tag{2}$$

$$\Sigma^3\Sigma_{kc} = \Sigma_c\left(\frac{\Sigma^2}{2} - k\right)\left(\frac{\Sigma_k\Sigma^2}{2} + k\Sigma_k - \Sigma\right), \tag{3}$$

where subscripts denote partial derivatives $\Sigma_c = \frac{\partial}{\partial c}\Sigma$ and $\Sigma_k = \frac{\partial}{\partial k}\Sigma$. Using (1), (2), and (3) we design an algorithm to approximate $\Sigma_{BS}(k,c)$ with two-dimensional polynomials in variables (k,c). Our implementation improves the existing methods by increasing the speed and accuracy. We further demonstrate how (1–3) can lead to convexity results for derivatives of Σ. The method described here can also be applied to the Bachelier model.

[1]The Black-Scholes price function is strictly monotone in volatility.

[2]As defined in Section 4, $k = \log\frac{Ke^{-rT}}{S_0e^{-qT}}$ denotes the log-moneyness, and $c = \frac{C_0}{S_0e^{-qT}}$ denotes the normalized price.

2. Practitioner's Corner

The partial differential equation (1–3) makes it possible to design an algorithm for approximation of the implied volatility. The algorithm consists of two phases. The first is "precomputation" phase in which we construct the partition of the (k, c) plane and the polynomials. The second is "runtime" phase in which the implied volatility is repeatedly calculated for different choices of parameters from the market.[3]

In the precomputation phase we specify the precision ε, degree d for the polynomials, and choose the desired range for parameters k, c, and σ. Small choices for ε result in finer partitions and larger number of polynomials. Likewise, intervals for k, c, and σ that include values close to 0 lead to more complex partitions.

We will first present the brief outlines of the two phases of the algorithm. The full details are developed later in the text.

Algorithm 1 Precomputation Phase

Input

1: ε = desired precision
2: $[k_{\min}, k_{\max}]$ = domain for the log-moneyness k
3: $[c_{\min}, c_{\max}]$ = domain for the normalized option price c
4: $[\sigma_{\min}, \sigma_{\max}]$ = domain for integrated volatility
5: d = required degree for polynomials

Output

1: $\{\kappa_0, \kappa_1, \ldots, \kappa_m\}$ = partition of the segment $[k_{\min}, k_{\max}]$
2: $\{c_{i,0}, c_{i,1}, \ldots, c_{i,\max_i}\}$ = partitions of the segment $[c_{\min}, c_{\max}]$ for each $i \in \{0, 1, \ldots, m\}$
3: $P_{i,j}(k, c)$ = polynomials of two variables of degree d that approximate integrated volatility $\Sigma_{BS}(k, c)$ with precision ε on the interval $[\kappa_i, \kappa_{i+1}] \times [c_{i,j}, c_{i,j+1}]$ for each $i \in \{0, 1, 2 \ldots, m\}$ and each $j \in \{0, 1, \ldots, \max_i\}$.

Code

```
1: function PrecomputationPhase
2:     {κ_0, κ_1, ..., κ_m} ← KAxisPartition (ε, d, k_min, k_max)
3:     for i ∈ {0, 1, 2, ..., m} do
4:         {c_{i,0}, c_{i,1}, ..., c_{i,max_i}} ← CAxisPartition (ε, d, c_min, c_max, κ_i)
5:         for j ∈ {0, 1, 2, ..., max_i} do
6:             P_{i,j} ← CreatePolynomial (d, κ_i, c_j)
7:     return {κ_i, c_{i,j}, P_{i,j}}
```

The functions KAxisPartition, CAxisPartition, and CreatePolynomial are based on the partial differential equation (1–3) and they are implemented in Section 7. We will now outline the data structures that are constructed after Algorithm 1 is executed. The numbers $\{\kappa_i, c_{i,j}\}$ determine the partition of the region $[k_{\min}, k_{\max}] \times [c_{\min}, c_{\max}]$ of the (k, c) plane. A depiction of such partition is shown in Figure 1. In addition, for each pair (i, j) we have a polynomial

$$P_{i,j} : [\kappa_i, \kappa_{i+1}] \times [c_{i,j}, c_{i,j+1}] \to \mathbb{R}$$

[3]This structure is identical with the one recently proposed in [18].

that approximates the integrated volatility $(k, c) \mapsto \Sigma_{BS}$ with precision ε.

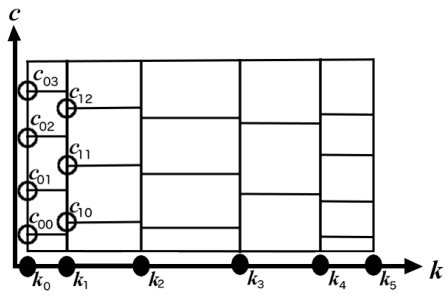

FIGURE 1. Partition of the (k, c) plane.

The runtime phase of the algorithm (see Algorithm 2) starts by uploading the partition and the polynomials that are generated by Algorithm 1. Then we can repeatedly receive the input parameters from the market and approximate the integrated volatility by identifying the appropriate polynomial and evaluating it.[4]

Algorithm 2 Runtime Phase

Input

1: $(S_0, K, C_{BS}, r, q, T)$ = (asset spot price, strike, market price of the option, interest rate, dividend rate, expiration)

Output

1: σ_{BS} = implied volatility

Code

1: **function** RUNTIMEPHASE
2: $\{\kappa_i, c_{i,j}\} \leftarrow$ LoadPartition$(\varepsilon, k_{\min}, k_{\max}, c_{\min}, c_{\max}, \sigma_{\min}, \sigma_{\max}, d)$
3: $\{P_{i,j}\} \leftarrow$ LoadPolynomials$(\varepsilon, k_{\min}, k_{\max}, c_{\min}, c_{\max}, \sigma_{\min}, \sigma_{\max}, d)$
4: $(k, c) \leftarrow \left(\log \frac{Ke^{-rT}}{S_0 e^{-qT}}, \frac{C_0}{S_0 e^{-qT}}\right)$
5: **if** $k < k_{\min}$ **then**
6: $k \leftarrow 1.01 \cdot 10^{-7}$
7: $(i, j) \leftarrow$ the pair of indices such that $(k, c) \in [\kappa_i, \kappa_{i+1}] \times [c_{i,j}, c_{i,j+1}]$
8: $\Sigma_{BS} \leftarrow P_{i,j}(k, c)$
9: **return** $\frac{\Sigma_{BS}}{\sqrt{T}}$

Our implementation of the above algorithm in C++ and OpenCL is available as an open source [39].

[4]Because of the divergence of our approximation at $k = 0$ as explained later in the text, in lines 5 and 6 of Algorithm 2 the small values of k are replaced by $1.01 \cdot 10^{-7}$. The chosen overall precision error $\varepsilon = 10^{-7}$ is unaffected.

3. Overview of the results from the literature

3.1. **Existing methods.** Originally suggested by [27], the implied volatility has been omnipresent in financial industry as it is common practice on trading desks for the implied volatility rather than the option price to be quoted. This practice allows option market makers to compare option prices across different asset classes. As stated in [2, 11], very accurate real-time calculations of implied volatility for millions of option prices are often required for high-frequency trading. Moreover, intraday recalibrations of more sophisticated stochastic volatility models that have been introduced and studied since the Black-Scholes model (see [14], for example) are typically performed by minimizing the difference of the corresponding implied volatilities, rather than the difference of model and market prices, across a number of model parameters. Since the inversion of the Black-Scholes formula cannot be done in a closed form (see [17]), numerous numerical procedures, as well as analytical approximations have been devised in the past. First numerical procedures were based on *iterative root finding algorithms* such as the bisection, Newton–Raphson, or Dekker–Brent method. While Newton–Raphson method has quadratic convergence (when it does converge, see [37]), it has pitfalls such as divisions by the option vegas that can be extremely small for away-from-ATM options [22]. Most commercial software (e.g. Matlab) uses the considerably slower Dekker–Brent algorithm that is guaranteed to converge. Although these methods work well for single options, these numerical solvers are too slow for real-time applications. To overcome this problem, Jäckel [22] exploits the limit behavior of the normalized option price to provide a smarter initial guess, which reduces the number of iterations in the Newton–Raphson algorithm, modified according to Halley's method. In [23], this approach is further improved using four rational function branches for the initial guess and the order four convergence Householder method for the iteration. In spite of being the state-of-the-art numerical solver for implied volatility applicable to any domain of input variables (see [29]), Jäckel's solver comes with the burden of a relatively complex implementation that makes it slower than some other methods reviewed below.

Non-iterative approximations to implied volatility are a popular alternative to the above-mentioned iterative solvers, as they provide closed-form expressions that are fast and easy to implement and maintain, and that can be eventually used for other gradient-based optimization routines. In the early works of Brenner and Subrahmanyan [5], Chance [7], Corrado and Miller [10], Li [31], and others (see [8, 11, 41], for review, discussion, and comparison), as well as in the more recent work of Lorig et al. [34] (that applies to more general local-stochastic volatility models), the analytical approach was almost exclusively based on *Taylor series expansions* of the call price around ATM (ATM) to a certain order, and as such, it suffers from a serious lack of precision for options that are more than 10% away from ATM, yielding these methods impractical. Over the past two decades, there has been a great amount of focus on obtaining asymptotic expansions of implied volatilities in extreme regions where numerical schemes typically become less efficient (see, for example, [15] for small maturity, [25] for large-time behavior, and [30] for extreme strikes). In principle, one can derive an approximation, containing no special functions and requiring no numerical integration, for the implied volatility in any stochastic volatility model whose characteristic function is available in closed form [26], by truncating its expansion with respect to some small parameter.

However, the resulting volatility expansion is accurate only around ATM, and there is no reason why it should be so in the tails.

With the work of Li [32], economized *rational approximations* for implied volatility have entered the picture, obtaining a better fit over a wider moneyness region, while losing some of the goodness of fit near ATM. Li used a fine mesh of about one million options to densely populate the 2-dimensional domain of approximation in log-moneyness and integrated volatility, and subsequently employed the downhill simplex search method to minimize an objective function, obtaining a single rational function of log-moneyness k and normalized price c with about 30 parameters, that can invert one million options within a few seconds. The domain $D_{[32]}$ covered by this approximation is fairly large, with k ranging from -0.5 and 0.5, and integrated volatility ranging from 0 to 1, while the uniform error of 10^{-3} order in the integrated volatility is claimed numerically.[5] For comparison, previously described Jäckel's solver [23] achieves the error of 10^{-14} order in the same domain, for a price of additional time (slower by roughly a factor of 10), complexity and maintenance required. One should keep in mind that Li's approximation is tailored to a specific fine mesh of options that covers only about 80% of the whole range of traded options in many cases (see [18]), excluding many options on underlyings with very low or high volatilities, as well as deep in-the-money (ITM) and out-of-the-money (OTM) options. Li's rational approximation was improved on $D_{[32]}$ in [43] and [45]. Pistorius and Stolte [43] use an iterative method of Press et al. [44] (Section 5.13) to obtain, for 105 different values of k, (univariate) Chebyshev rational functions of $\sqrt{c}$ with varying degrees (7 to 9), and eventually proceed with cubic splines to interpolate in between, reducing the maximum absolute error to the order of 10^{-5}. Although, much like with [32], all approximations are done once and stored to the hard-drive, so few calculations are required during the runtime, this improvement in error comes with increased storage and no significant improvement in speed, while still struggling where most methods do: with deep ITM/OTM options as well as options with short maturity. Salazar Celis in [45] works on a Chebyshev grid that is structured in k-direction, scattered in c-direction, and most importantly denser in the corners of $D_{[32]}$, and uses singular value decomposition to solve the highly non-linear optimization problem of finding a univariate rational function of $\sqrt{c}$, with weights that are polynomial in k, hence obtaining a parametrized barycentric approximation that is (unlike [43]) truly bivariate rational. With no improvement in speed over [32] reported, the maximum absolute error is of the order of 10^{-4} on $D_{[32]}$, albeit requiring storage of only 70 coefficients.

Glau et al. [18] used Jäckel's approximation [23] at the gridpoints of a large 1000×1000 Chebyshev grid in the (k, c)-space, and a *chebfun* Matlab package for interpolation in between, thus obtaining a bivariate Chebyshev *polynomial* approximation for implied volatility with subexponential rate of convergence. On domain $D_{[32]}$ it achieves the error of order $10^{-5}/10^{-8}/10^{-11}$. Their Matlab implementations are roughly 4/3/2 times faster (respectively) than the algorithm of Jäckel [23], and 4/5/6 times slower than the algorithm of Li [32] (whose inferior error is of order 10^{-3} on the same domain). However, like [23], and unlike [32], this approach is applicable on a much wider approximation domain $D_{[18]}$, with k ranging from -5 and 5, and integrated volatility ranging from 0 to 6, that completely covers

[5] It should be noted that one still needs to divide by $\sqrt{T}$ to obtain the implied volatility, making the actual error significantly bigger for short maturities.

the market data. In comparison to Jäckel's method [23] whose error is of order 10^{-13} on $D_{[18]}$, the approximation of [18] has the error of order $10^{-5}/10^{-8}/10^{-10}$ in roughly $4/3/2$ times shorter running time, respectively. It is worth noting that the Chebyshev method from [18] is the first to use polynomials to approximate the implied volatility. Their method is similar to ours in that it uses the domain partitioning in pre-computation phase leaving the evaluation of polynomials for the runtime phase. There are several caveats with their method, though: their approximation polynomials have significantly larger degrees than ours, degrees vary across different domains (unlike ours), and the data transformations before evaluations of Chebyshev polynomials are computationally expensive as they feature multiple logarithms, exponential functions, and square roots.

In [33], Li and Lee introduced a new successive over-relaxation (SOR) *iterative* method to compute the implied volatility, that is applicable on $D_{[18]}$, and has a quadratic order of convergence of Newton-Raphson algorithm and the robustness of the Dekker-Brent algorithm. With just five iterations, it achieves the uniform error of order 10^{-13} in the integrated volatility, and manages to invert one million options in about 10 seconds. Moreover, its accuracy is reported (see [47]) to be better than that of [23], quite possibly due to the fact that already the generation of the initial guess in Jäckel's algorithm relies on rational cubic interpolations and transformations that are highly sensitive in terms of the accuracy of the error function and the inverse of the normal cumulative distribution function (CDF).

Finally, in a series of papers [38, 48, 49], the authors derived a simple closed-form approximation formula for the implied volatility based on the *Pólya-type approximations* of the normal CDF. The relative error of this approximation is uniformly bounded for options with any moneyness and with arbitrary large or small option maturities and volatilities, including for long dated options and options on highly volatile underlying assets. It is reported in [29] to be more robust than the rational approximation of Li [33]. For ATM options (i.e. when $k = 0$), the approximation is even more precise, having relative error of order 10^{-6} for all options with integrated volatility less than 2. Furthermore, Gatheral and the authors [16] proved it is actually a lower bound on the implied volatility over a large range of strikes and option prices. Moreover, combined with the variational principle of Tehranchi [51], they obtained tight uniform bounds on the implied volatility that can be used as initial guesses to effectively speed up the bisection method, the most basic iterative root finding algorithm still used by many practitioners. When the SOR method, reviewed in the previous paragraph, uses this approximation as an initial guess, it has an implementation that is faster than Jäckel's method for most of the input values [29, 47].

While we are fully aware that comparisons of approximation algorithms and formulae, described above in some detail, may depend on the performance measures (e.g. maximum absolute error, relative error, speed, storage, maintenance), implementations, frameworks, libraries, languages and platforms used, the general strategy for inverting millions of option prices for Black-Scholes implied volatility in practice seems to be to choose the SOR method of Li and Lee [33], if one opts for optimal speed, the solver of Jäckel [23] if one opts for optimal accuracy, or the closed-form approximation of [49], if one opts for simplicity. Therefore, it is safe to conclude that the new algorithm for approximating the implied volatility derived in this paper should be compared to the two state-of-the-art numerical solvers for

implied volatility of [23] and [33] (the latter implemented with the approximation of [49] as the initial guess), that are available as open source.

3.2. **PDE-based method.** Our new numerical procedure is relatively simple in comparison with previous methods. It is concise and analytic, being based on partial differential equations (PDEs) for the implied volatility and appropriate recursions derived to calculate their power series solutions. Although using power series to solve differential equations is a classical method that can be traced back to Frobenius [13] or even Euler (see also [40, 42]), the literature is scarce when it comes to differential equations satisfied by the implied volatility; the singular occurrence being the result of Berestycki et al. [3] showing that, in a rather general class of stochastic volatility models, implied volatility in the small-maturity limit can be characterized as a viscosity solution to a first order non-linear eikonal PDE. A method similar to ours was used in [50] for establishing ordinary differential equations (ODEs) for quantile functions of common probability distributions, which in turn yields immediate applications in modern methods of Monte–Carlo simulation, techniques based on low-discrepancy sequences, as well as copula methods. In particular, the ODE for the quantile of the normal CDF is a special case of the PDE for the implied volatility in the Black–Scholes model, namely, equation (4) from [50] turns out to be a special case of equation (1) in Theorem 1 when $k = 0$. It is worth noting that a recent result by Xia and Cui [54] provides one-dimensional power series expansion for the implied volatility for a fixed strike. In fact, their Theorem 4 after proper transformations is actually equivalent to our power series (27) in variable c (with k constant).

After obtaining the PDEs for the implied volatility, a bird's–eye view of our numerical scheme sees the entire (k, c)-domain partitioned into rectangular cells and on each cell uses a (two-dimensional) polynomial approximation around its bottom-left corner, with coefficients obtained recursively from the PDEs.[6] Our partition is specifically tailored to the application at hand, yielding additional accuracy over methods that use uniform, or even Chebyshev grids. Namely, the precision of the polynomials depends on the point the expansion is made around and, consequently, the partition of the domain is driven to be non-uniform, as it can clearly be seen in Figure 1, where the grid cells are much smaller and abundant for small values of k and/or when c is close to 0 or 1; these values corresponding to the ATM options, and deep OTM/ITM options, respectively. The precomputation phase of the algorithm consists of preparing the partition and the polynomials for given precision ε. After this preparation is completed, the evaluation of the polynomials is fast for any given set of inputs. The full details of our implementation are provided in Section 7.

Our procedure for approximating the implied volatility provides a combination of efficiency, accuracy, and robustness (in a sense that the precision and the domain of input variables can be adjusted for the application at hand). It covers a large domain of input variables, as it is able to deal with small and large (absolute) values of log-moneyness k (that is, deep ITM/OTM options), short times to maturity, large values of underlying, as well as options with very high/low volatilities. We will illustrate our method on the domain where log-strike k and relative price c satisfy $k \in \left[10^{-6}, 5\right]$ and $c \in \left[10^{-6}, 0.997\right]$. The range for the integrated volatility

[6]The use of local polynomial expansion around many points yields the apparent precision of our algorithm, and is reminiscent of their use in Hairer's theory of regularity structures [21].

that is covered by the method is $\sigma\sqrt{T} \in \left[10^{-6}, 7.3\right]$. For comparison, the domain $D_{[32]}$, covered by some of the existing methods, required that $\frac{k}{\sigma\sqrt{T}}$ is smaller than 2. Our method allows the quantity $\frac{k}{\sigma\sqrt{T}}$ to range over the interval $\left[2.6 \cdot 10^{-6}, 4.9\right]$.

For the chosen precision level of 10^{-7}, it is sufficiently accurate for practical purposes and highly efficient in terms of absolute error and speed with large data sets, delivering precise real-time evaluations of implied volatility. For example, the running time of our implementation to invert one million randomly generated option prices for implied volatility is about three times shorter than that of [23] and [33] on a single processor. Our algorithm is easy to implement and maintain, and, unlike the competing routines of [18, 23, 33, 43, 45], it is completely independent of more involved numerical recipes, such as splines, rational approximation packages, singular value decomposition, or transformations/objective functions that involve the error function or the inverse of the normal CDF. Indeed, in a commercial environment, it is of little significance what may be implemented in, say Matlab, while our implementation can be easily extracted and ported to the respectively relevant industrial system or analytics library. With maximum absolute error of 10^{-7}, the number of calculated constants is 2.5 million in the precomputation phase that needs to be executed only once. It is also very robust, as both precision and the domain of input variables can be easily improved/extended, alas for the price of slower running time and additional memory to store partitions and polynomials. Another main advantage of our numerical scheme lies in a fewer number of conditional statements and branchings/domain splittings. Although our approximation diverges for $k = 0$ (that corresponds to the ATM options), simply replacing $k = 0$ with $k = 1.01 \cdot 10^{-7}$ handles the ATM case and maintains the same absolute error of 10^{-7}.

The gain in speed can be made to be even more substantial. Our OpenCL implementation on p3.2xlarge instance on Amazon Web Services to invert one million randomly generated option prices for implied volatility is five times faster than when the algorithm is executed on an eight-core CPU. Since graphic cards consist of numerous processing elements that are performing optimally if all pipelines are equally loaded, numerous branchings/domain splittings in the competing routines of [23, 33] make them less suitable for parallelization. In addition, note that our partition of the domain is done with line segments parallel to the coordinate axes (see Figure 1). This feature allowed us to use a simple indexing algorithm of $O(1)$ time complexity and avoid look-up bottleneck for parallelization purposes. If one were to circumvent the coordinate transformations performed before evaluations of Chebyshev polynomials (see $\phi_{i,x}(c)$ on pages 10–13 of [18]), the Chebyshev method would also be amenable to an efficient indexing algorithm of $O(1)$ time complexity.

3.3. **Organization of the paper.** Section 4 introduces the necessary notation and defines the inverse problem of finding the implied volatility in the Black–Scholes model after the standard dimension reduction. Section 5 contains the proof of equations (1–3), stated as Theorem 1. In Section 6 we use equations (1–3) to obtain recursive formulas for two-dimensional polynomial approximation for the implied volatility. The details of the implementation are presented in Section 7. The speed and accuracy of our method is compared to that of other numerical schemes in Section 8. Our method of finding a differential equation satisfied by the implied volatility (after an appropriate dimension reduction) appears to be

general enough to be applicable in models other than the Black–Scholes model. In Section 9, we derive an efficient and accurate numerical scheme for the calculation of implied Bachelier volatility that surpasses the rational approximation of [9] and provides a more transparent, albeit less accurate, alternative to recent machine-precision approximations of [24, 28]. Since inverting for Bachelier volatility is a one-dimensional problem, it may be helpful for the reader to first go through Section 9 before delving into more technical implementations for Black–Scholes implied volatility presented in Sections 6 and 7. Section 10 discusses the drawbacks of our method and provides examples of how it can be of further use for approximations and derivations of theoretical results. We conclude the paper with remarks in Section 11.

4. Implied volatility in Black-Scholes model

In what follows, we adopt the approach and notation of [16] (see also [32, 45]). Let S_0 be the current stock price, σ_{BS} the volatility of the stock, r the risk-free interest rate and q the dividend rate. Let C_0 be the price of a plain vanilla European call option at time 0 with expiration date T and strike price K. Then the Black–Scholes formula gives C_0 in closed form as follows:

$$C_0\left(S_0, K, \sigma_{BS}, r, q, T\right) \quad = \quad S_0 e^{-qT} C_{BS}\left[\log \frac{K e^{-rT}}{S_0 e^{-qT}}, \sigma_{BS}\sqrt{T}\right],$$

where the function $C_{BS} : \mathbb{R} \times [0, +\infty) \to [0, 1)$ is defined as

$$C_{BS}(k, \sigma) \quad = \quad N\left(-\frac{k}{\sigma} + \frac{\sigma}{2}\right) - e^k N\left(-\frac{k}{\sigma} - \frac{\sigma}{2}\right). \tag{4}$$

Finding the implied volatility reduces to the following problem

For given c and k find σ that solves $C_{BS}(k, \sigma) = c$.

Definition 1. *The function $\Sigma_{BS}(k, c)$ is the inverse of $C_{BS}(k, \sigma)$. In other words*

$$\Sigma_{BS}(k, \cdot) : \left[\left(1 - e^k\right)^+, 1\right) \to [0, +\infty)$$

is given by

$$\sigma = \Sigma_{BS}(k, c) \iff c = C_{BS}(k, \sigma). \tag{5}$$

In practice one observes the values C_0, K, r, q, and T, and evaluates

$$\text{the normalized call option price } c \quad = \quad \frac{C_0}{S_0 e^{-qT}}, \tag{6}$$

$$\text{the log-moneyness } k \quad = \quad \log \frac{K e^{-rT}}{S_0 e^{-qT}}. \tag{7}$$

After obtaining the integrated volatility $\sigma = \Sigma_{BS}(k, c)$, the volatility σ_{BS} can be recovered from σ using

$$\sigma_{BS} \quad = \quad \frac{\sigma}{\sqrt{T}}. \tag{8}$$

It is known that one merely needs to focus on call prices due to the put-call parity

$$C_{BS}(k, \sigma) = P_{BS}(k, \sigma) + 1 - e^k, \tag{9}$$

where the normalized put price $P_{BS}(k, \sigma)$ is defined in analogy to (4).

Our domain of approximation can be restricted to $k \geq 0$, due to the following duality

$$C_{BS}(-k,\sigma) = e^{-k}C_{BS}(k,\sigma) + 1 - e^{-k}, \tag{10}$$

which implies

$$\Sigma_{BS}(-k,c) = \Sigma_{BS}(k, e^{-k}c + 1 - e^{-k}). \tag{11}$$

5. Piecewise approximation of the implied volatility

In this section we often omit the subscript in $\Sigma_{BS}(k,c)$ and write only $\Sigma(k,c)$ instead. For clarity of exposition, we again write equations (1), (2), and (3), now stated as our main theorem.

Theorem 1. *The implied volatility $\Sigma(k,c)$ satisfies the following three partial differential equations*

$$\Sigma^3\Sigma_{cc} = \Sigma_c^2\left(\frac{\Sigma^4}{4} - k^2\right), \tag{1}$$

$$\Sigma^3\Sigma_{kk} = \left(\frac{\Sigma^2\Sigma_k}{2} + \Sigma - k\Sigma_k\right)\left(\frac{\Sigma^2\Sigma_k}{2} - \Sigma + k\Sigma_k\right), \quad \textit{and} \tag{2}$$

$$\Sigma^3\Sigma_{kc} = \Sigma_c\left(\frac{\Sigma^2}{2} - k\right)\left(\frac{\Sigma_k\Sigma^2}{2} + k\Sigma_k - \Sigma\right). \tag{3}$$

Proof. We will start from the relation $C_{BS}(k, \Sigma_{BS}(k,c)) = c$ that can be written in an equivalent form

$$c = N\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right) - e^k N\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right). \tag{12}$$

The derivative of $N(z)$ satisfies $N'(z) = \frac{1}{\sqrt{2\pi}}e^{-\frac{z^2}{2}}$. Therefore we obtain the following two relations

$$N'\left(-\frac{k}{y} + \frac{y}{2}\right) = e^k N'\left(-\frac{k}{y} - \frac{y}{2}\right) \tag{13}$$

$$N''(z) = -zN'(z). \tag{14}$$

Taking partial derivatives with respect to c of both sides of (12) gives us

$$\begin{aligned} 1 = {} & e^k N'\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right)\left(\frac{\partial}{\partial c}\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right)\right. \\ & \left. -\frac{\partial}{\partial c}\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right)\right). \end{aligned}$$

After simplifying the right-hand side we obtain

$$1 = N'\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right)\cdot\Sigma_c(k,c). \tag{15}$$

The last equation can be re-written as

$$\Sigma_c(k,c) = \sqrt{2\pi}e^{\frac{1}{2}\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right)^2}. \tag{16}$$

Once we differentiate both sides of (15) with respect to c and use (14) we obtain equation (1).

We can obtain the derivative Σ_k by differentiating both sides of (12) with respect to k. Using (13) and (14) we obtain

$$\Sigma_k(k,c) = \frac{N\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right)}{N'\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right)}. \tag{17}$$

We can re-write (17) as

$$N'\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right)\Sigma_k(k,c) = N\left(-\frac{k}{\Sigma(k,c)} - \frac{\Sigma(k,c)}{2}\right). \tag{18}$$

Differentiating both sides with respect to k and using (14) we obtain (2).

Once we take the derivative of both sides of (15) with respect to k we obtain

$$\begin{aligned} 0 = {} & N''\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right)\cdot\Sigma_c(k,c)\cdot\frac{\partial}{\partial k}\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right) \\ & + N'\left(-\frac{k}{\Sigma(k,c)} + \frac{\Sigma(k,c)}{2}\right)\cdot\Sigma_{kc}(k,c). \end{aligned}$$

Using (14) we can further simplify the last equation and obtain (3).

□

6. Power series representation

In this section we will do the same as in the previous one – omit the subscript in $\Sigma_{BS}(k,c)$ and work with $\Sigma(k,c)$. We will use equations (1–3) to obtain the recursive relations allowing us to determine the two dimensional sequence $\alpha_{m,n} = \alpha_{m,n}(k_0,c_0)$ of real numbers such that

$$\Sigma(k,c) = \sum_{m,n=0}^{\infty}\alpha_{m,n}\left(c-c_0\right)^m\left(k-k_0\right)^n. \tag{19}$$

We will first use (1) to construct a recursion for the sequence $(\alpha_{m,0})_{m=0}^{\infty}$. Then we will use (2) to create a recursion for $(\alpha_{0,n})_{n=0}^{\infty}$. After finding the sequences $(\alpha_{m,0})_{m=0}^{\infty}$ and $(\alpha_{0,n})_{n=0}^{\infty}$ we will use (3) to get a recursive relation for $\alpha_{u,v}$ in terms of $\alpha_{u',v'}$ with $u' \le u$ and $v' \le v$. In other words, we will prove that $\alpha_{u,v}$ is a polynomial function of variables $\alpha_{m,n}$ whose indices are pairs of numbers (m,n) that belong to the set

$$S_{u,v} = \left\{(u',v') : 0 \le u' \le u, 0 \le v' \le v, (u',v') \ne (u,v)\right\}. \tag{20}$$

Let us first observe that the following identities hold for partial derivatives of $\Sigma(k,c)$

$$\Sigma_c(k,c) = \sum_{m,n=0}^{\infty}(m+1)\alpha_{m+1,n}\left(c-c_0\right)^m\left(k-k_0\right)^n, \tag{21}$$

$$\Sigma_k(k,c) = \sum_{m,n=0}^{\infty}(n+1)\alpha_{m,n+1}\left(c-c_0\right)^m\left(k-k_0\right)^n, \tag{22}$$

$$\Sigma_{kc}(k,c) = \sum_{m,n=0}^{\infty}(m+1)(n+1)\alpha_{m+1,n+1}\left(c-c_0\right)^m\left(k-k_0\right)^n. \tag{23}$$

6.1. **Calculation of $\alpha_{m,0}$.** Assume that we are given c_0, Σ_0, and k_0 such that $\Sigma_0 = \Sigma_{BS}(k_0, c_0)$. In this section the value $k = k_0$ is fixed and the function $c \mapsto \Sigma(k_0, c)$ depends only on one variable c. The coefficients in power series expansion of such function will form the sequence $(\alpha_{m,0})$. In order to simplify notation and emphasize that we are dealing with functions of a single variable we will use α^C to label this sequence. In other words, we define

$$\alpha_m^C = \alpha_{m,0}.$$

The first term satisfies $\alpha_0^C = \Sigma_0$. Together with (16) and (1) we obtain the equations for the first three terms of the sequence $\left(\alpha_m^C\right)_{m=0}^{\infty}$

$$\alpha_0^C = \Sigma_0, \tag{24}$$

$$\alpha_1^C = \sqrt{2\pi} \cdot e^{\frac{1}{2}\left(-\frac{k_0}{\Sigma_0}+\frac{\Sigma_0}{2}\right)^2}, \tag{25}$$

$$\alpha_2^C = \frac{1}{2} \cdot \frac{\left(\alpha_1^C\right)^2}{\left(\alpha_0^C\right)^3} \cdot \left(\frac{\left(\alpha_0^C\right)^4}{4} - k_0^2\right). \tag{26}$$

Placing $k = k_0$ in equation (19) and differentiating twice implies

$$\Sigma(k_0, c) = \sum_{m=0}^{\infty} \alpha_m^C (c - c_0)^m \tag{27}$$

$$\Sigma_c(k_0, c) = \sum_{m=0}^{\infty} (m+1)\alpha_{m+1}^C (c - c_0)^m \tag{28}$$

$$\Sigma_{cc}(k_0, c) = \sum_{m=0}^{\infty} (m+2)(m+1)\alpha_{m+2}^C (c - c_0)^m. \tag{29}$$

We will consider the sequences $\left(\alpha_m^{CD}\right)_{m=0}^{\infty}$ and $\left(\alpha_m^{CDD}\right)_{m=0}^{\infty}$ that correspond to the expansions of the first and second derivatives of $\Sigma(k_0, \cdot)$ in (28) and (29). The sequences are defined as

$$\alpha_m^{CD} = (m+1)\alpha_{m+1}^C, \tag{30}$$

$$\alpha_m^{CDD} = (m+2)(m+1)\alpha_{m+2}^C. \tag{31}$$

Assume now that $u \geq 2$ is fixed and that we know all of the terms α_i^C for $i \in \{0, 1, 2, \ldots, u\}$. The goal is to find a procedure to calculate α_{u+1}. The idea is to compare the coefficients of $(c - c_0)^{u-1}$ on both sides of (1). Let us recall the definition of the discrete convolution of sequences. For two sequences $(\mu_m)_{m=0}^{\infty}$ and $(\nu_m)_{m=0}^{\infty}$ their convolution $\mu * \nu$ is defined as the sequence $(\mu * \nu)_{m=0}^{\infty}$ that satisfies

$$(\mu * \nu)_m = \sum_{i=0}^{m} \mu_i \nu_{m-i}.$$

If the sequences μ and ν are generating the power series $M(x) = \sum_{i=0}^{\infty} \mu_i (x - x_0)^i$ and $N(x) = \sum_{i=0}^{\infty} \nu_i (x - x_0)^i$ then $\mu * \nu$ is generating the power series of the product $M(x) \cdot N(x)$.

Let us denote by L_{u-1}^C the coefficient on the left-hand side of (1) corresponding to $(c - c_0)^{u-1}$. Since the power series is the product of four power series $\Sigma \cdot \Sigma \cdot \Sigma \cdot \Sigma_{cc}$

the sequence of coefficients L^C is the convolution of four sequences. Three of them are equal to α^C while the fourth sequence is α^{CDD}. In other words,

$$L^C_{u-1} = \left(\alpha^C * \alpha^C * \alpha^C * \alpha^{CDD}\right)_{u-1}. \tag{32}$$

The convolution is commutative and associative and there is no need for brackets to specify the order of operations.

Only one term in L^C_{u-1} depends on α^C_{u+1}, while all other summands depend on elements from the set $\left\{\alpha^C_i : 0 \le i \le u\right\}$. According to (31) this particular term in L^C_{u-1} that depends on α^C_{u+1} is obtained by multiplying lowest order terms from the three sequences α^C and the highest order term from the sequence α^{CDD}. Hence, it must be equal to

$$\left(\alpha^C_0\right)^3 \cdot \alpha^{CDD}_{u-1} = \Sigma_0^3 \cdot (u+1) \cdot u \cdot \alpha^C_{u+1}.$$

Let us denote by $L^{C,\text{lower}}_{u-1}$ the portion of the summation in L^C_{u-1} that is obtained using only elements of the sequence α^C whose indices are smaller than or equal to u. Using this notation we can write

$$L^C_{u-1} \quad = \quad \Sigma_0^3 \cdot (u+1) \cdot u \cdot \alpha^C_{u+1} + L^{C,\text{lower}}_{u-1}.$$

The coefficient of $(c-c_0)^{u-1}$ on the right-hand side of (1) satisfies

$$R^C_{u-1} \quad = \quad \left(\alpha^{CD} * \alpha^{CD} * \left(\frac{1}{4}\alpha^C * \alpha^C * \alpha^C * \alpha^C - k_0^2\right)\right)_{u-1}.$$

Notice that R^C_{u-1} is calculated using only the terms from the set $\left\{\alpha^C_i : 0 \le i \le u\right\}$. Therefore we obtain the recursion

$$\alpha^C_{u+1} \quad = \quad \frac{R^C_{u-1} - L^{C,\text{lower}}_{u-1}}{(u+1) \cdot u \cdot \Sigma_0^3}. \tag{33}$$

6.2. **Calculation of $\alpha_{0,n}$.** Assume that we are given c_0, Σ_0, and k_0 such that $\Sigma_0 = \Sigma_{BS}(k_0, c_0)$. We will find the expansion of the single-variable function $k \mapsto \Sigma(k, c_0)$ around k_0. Let us denote $\alpha^K_n = \alpha_{0,n}$. We proceed in a similar way as in Subsection 6.1. The equation (19) with $c = c_0$ becomes

$$\Sigma(k, c_0) \quad = \quad \sum_{n=0}^{\infty} \alpha^K_n (k-k_0)^n \tag{34}$$

$$\Sigma_k(k, c_0) \quad = \quad \sum_{n=0}^{\infty} (n+1)\alpha^K_{n+1} (k-k_0)^n \tag{35}$$

$$\Sigma_{kk}(k, c_0) \quad = \quad \sum_{n=0}^{\infty} (n+2)(n+1)\alpha^K_{n+2} (k-k_0)^n. \tag{36}$$

One has to take care of the appearance of the variable k on the right-hand side of (2) which is no longer a constant. We re-write (2) as

$$\begin{aligned} \Sigma^3\Sigma_{kk} \quad = \quad & \left(\frac{\Sigma^2\Sigma_k}{2} + \Sigma - k_0\Sigma_k - (k-k_0)\Sigma_k\right) \\ & \cdot \left(\frac{\Sigma^2\Sigma_k}{2} - \Sigma + k_0\Sigma_k + (k-k_0)\Sigma_k\right). \end{aligned} \tag{37}$$

As in Subsection 6.1 we introduce the sequences $\alpha_n^{KD} = (n+1)\alpha_{n+1}^K$ and $\alpha_n^{KDD} = (n+2)(n+1)\alpha_{n+2}^K$. These two sequences are generating the power series expansions for the functions $\Sigma_k(\cdot, c_0)$ and $\Sigma_{kk}(\cdot, c_0)$, respectively. We also need the sequence that generates the power series for $k \mapsto (k-k_0)\Sigma_k(k, c_0)$. This sequence is obtained as a shift to the right of the sequence α^{KD}. We will denote it by $\alpha^{KD\rightarrow}$. The terms of this sequence must satisfy

$$\alpha_n^{KD\rightarrow} = \begin{cases} 0, & \text{if } n = 0 \\ \alpha_{n-1}^{KD}, & \text{if } n > 0. \end{cases} \tag{38}$$

The formulas for the initial members α_0^K, α_1^K, and α_2^K are calculated using (17) and (2). They are given by

$$\alpha_0^K = \Sigma_0, \tag{39}$$

$$\alpha_1^K = \frac{N\left(-\frac{k_0}{\Sigma_0} - \frac{\Sigma_0}{2}\right)}{N'\left(-\frac{k_0}{\Sigma_0} - \frac{\Sigma_0}{2}\right)}, \tag{40}$$

$$\alpha_2^K = \frac{\left(\frac{\Sigma_0^2\alpha_1^K}{2} + \Sigma_0 - k_0\alpha_1^K\right) \cdot \left(\frac{\Sigma_0^2\alpha_1^K}{2} - \Sigma_0 + k_0\alpha_1^K\right)}{2\Sigma_0^3}. \tag{41}$$

Having calculated α_0^K, α_1^K, $\ldots$, α_v^K for some $v \geq 2$ we obtain

$$\alpha_{v+1}^K = \frac{R_{v-1}^K - L_{v-1}^{K,\text{lower}}}{v(v+1)\Sigma_0^3}, \quad \text{where} \tag{42}$$

$$R_{v-1}^K = \left(\left(\frac{1}{2}\alpha^K * \alpha^K * \alpha^{KD} + \alpha^K - k_0\alpha^{KD} - \alpha^{KD\rightarrow}\right) * \left(\frac{1}{2}\alpha^K * \alpha^K * \alpha^{KD} - \alpha^K + k_0\alpha^{KD} + \alpha^{KD\rightarrow}\right)\right)_{v-1}, \tag{43}$$

$$L_{v-1}^{K,\text{lower}} = \left(\alpha^K * \alpha^K * \alpha^K * \alpha^{KDD}\right)_{v-1} - \Sigma_0^3 \cdot (v+1) \cdot v \cdot \alpha_{v+1}^K. \tag{44}$$

Here R_{v-1}^K and $L_{v-1}^{K,\text{lower}}$ are analogous to the corresponding R_{u-1}^C and $L_{u-1}^{C,\text{lower}}$ in Subsection 6.1, and a reasoning similar to the one in the paragraph after (32) implies that R_{v-1}^K and $L_{v-1}^{K,\text{lower}}$ are calculated using only the terms from the set $\left\{\alpha_i^K : 0 \leq i \leq v\right\}$.

6.3. **Calculation of $\alpha_{m,n}$.** As before, we assume that we are given k_0, c_0, and Σ_0 that already satisfy $\Sigma_0 = \Sigma_{BS}(k_0, c_0)$. We will first recall the basics of two-dimensional convolution. Assume that $(\mu_{m,n})_{m,n=0}^{\infty}$ and $(\nu_{m,n})_{m,n=0}^{\infty}$ are two two-dimensional sequences. Their convolution $\eta = \mu * \nu$ is defined as a new two-dimensional sequence $(\eta_{m,n})_{m,n=0}^{\infty}$ in the following way

$$\eta_{m,n} = \sum_{i+j=m, p+q=n} \mu_{i,p}\nu_{j,q}, \tag{45}$$

where the indices i, j, p, and q are non-negative integers. If the functions A and B have power series representations

$$A(x,y) = \sum \mu_{m,n}(x-x_0)^m(y-y_0)^n \quad \text{and}$$
$$B(x,y) = \sum \nu_{m,n}(x-x_0)^m(y-y_0)^n,$$

then their product satisfies

$$A(x,y)B(x,y) = \sum_{m,n=0}^{\infty} (\mu * \nu)_{m,n} (x-x_0)^m (y-y_0)^n .$$

Equation (3) contains a variable k. Unlike k_0, the quantity k is not a constant, so we will re-write the equation in the form

$$\begin{aligned} \Sigma^3 \Sigma_{kc} &= \Sigma_c \left(-k_0 - (k-k_0) + \frac{\Sigma^2}{2} \right) \cdot \\ &\quad \cdot \left(-\Sigma + k_0 \Sigma_k + (k-k_0)\Sigma_k + \frac{\Sigma_k \Sigma^2}{2} \right). \end{aligned} \tag{46}$$

Let us define the sequences $\alpha^{\partial C}$, $\alpha^{\partial K}$, $\alpha^{\partial C \partial K}$, $\alpha^{\partial K \rightarrow}$ to correspond to the functions Σ_c, Σ_k, Σ_{ck}, and $(k-k_0)\Sigma_k$, respectively. As it can be seen from the equations (21), (22), and (23), the formal definitions of these sequences can be written as

$$\begin{aligned} \alpha^{\partial C}_{m,n} &= (m+1)\alpha_{m+1,n}, \\ \alpha^{\partial K}_{m,n} &= (n+1)\alpha_{m,n+1}, \\ \alpha^{\partial C \partial K}_{m,n} &= (m+1)(n+1)\alpha_{m+1,n+1}, \\ \alpha^{\partial K \rightarrow}_{m,n} &= \begin{cases} 0, & \text{if } n = 0 \\ \alpha^{\partial K}_{m,n-1}, & \text{if } n > 0. \end{cases} \end{aligned}$$

Assume that $u \geq 1$ and $v \geq 1$ and that we know all the terms of $\alpha_{m,n}$ for which $0 \leq m \leq u$, $0 \leq n \leq v$ and $(m,n) \neq (u,v)$. In (20) we introduced the notation $S_{u,v}$ for the set of such pairs of integers (m,n). We will express $\alpha_{u,v}$ in terms of $\alpha_{u',v'}$ for $(u',v') \in S_{u,v}$ which are previously calculated. We do this by comparing the coefficients of $(c-c_0)^{u-1}(k-k_0)^{v-1}$ on both sides of (46).

Let us introduce the sequences $(\beta_{m,n})_{m,n=0}^{\infty}$, $(\gamma_{m,n})_{m,n=0}^{\infty}$, and $(\delta_{m,n})_{m,n=0}^{\infty}$ for which the following hold

$$-k_0 - (k-k_0) + \frac{\Sigma^2}{2} = \sum_{m,n=0}^{\infty} \beta_{m,n} (c-c_0)^m (k-k_0)^n , \tag{47}$$

$$\Sigma^3 = \sum_{m,n=0}^{\infty} \gamma_{m,n} (c-c_0)^m (k-k_0)^n , \tag{48}$$

$$-\Sigma + k_0\Sigma_k + (k-k_0)\Sigma_k + \frac{\Sigma_k \Sigma^2}{2} = \sum_{m,n=0}^{\infty} \delta_{m,n} (c-c_0)^m (k-k_0)^n . \tag{49}$$

We have that

$$\beta_{m,n} = \begin{cases} \frac{\Sigma_0^2}{2} - k_0, & \text{if } m = n = 0, \\ \frac{1}{2}(\alpha * \alpha)_{0,1} - 1, & \text{if } m = 0,\ n = 1, \\ \frac{1}{2}(\alpha * \alpha)_{m,n}, & \text{otherwise,} \end{cases} \tag{50}$$

$$\gamma = \alpha * \alpha * \alpha, \tag{51}$$

$$\delta = \frac{1}{2}\left(\alpha * \alpha * \alpha^{\partial K}\right) + \alpha^{\partial K \rightarrow} + k_0 \alpha^{\partial K} - \alpha. \tag{52}$$

Let us denote by $L_{u-1,v-1}$ and $R_{u-1,v-1}$ the coefficients on the left and right-hand side of (46) that correspond to the term $(c-c_0)^{u-1}(k-k_0)^{v-1}$. These coefficients must be equal. Since both sides of equations are products of two-dimensional polynomials the coefficients $L_{u-1,v-1}$ and $R_{u-1,v-1}$ can be obtained using convolutions. The coefficient $R_{u-1,v-1}$ satisfies

$$R_{u-1,v-1} \quad = \quad \left(\alpha^{\partial C} * \beta * \delta\right)_{u-1,v-1}, \tag{53}$$

hence it can be calculated using only the terms $\alpha_{u',v'}$ for $(u',v') \in S_{u,v}$. Similarly, $L_{u-1,v-1}$ satisfies

$$L_{u-1,v-1} \quad = \quad \left(\alpha^{\partial C \partial K} * \gamma\right)_{u-1,v-1}. \tag{54}$$

One of the terms in $L_{u-1,v-1}$ is $\gamma_{0,0} \cdot \alpha^{\partial C \partial K}_{u-1,v-1} = \alpha^3_{0,0} \cdot uv \cdot \alpha_{u,v}$ while the remaining terms can be calculated using only the variables $\alpha_{u',v'}$ for $(u',v') \in S_{u,v}$. Let us denote by $L^{\text{lower}}_{u-1,v-1}$ the portion of $L_{u-1,v-1}$ that is expressed in terms of $\alpha_{u',v'}$ for $(u',v') \in S_{u,v}$. In other words,

$$L^{\text{lower}}_{u-1,v-1} = L_{u-1,v-1} - \alpha^3_{0,0} \cdot uv \cdot \alpha_{u,v}. \tag{55}$$

Then we have

$$\alpha_{u,v} \quad = \quad \frac{R_{u-1,v-1} - L^{\text{lower}}_{u-1,v-1}}{\Sigma^3_0 \cdot u \cdot v}. \tag{56}$$

The last equation gives us the recursive procedure for calculating the terms of the sequence α. The initial data are the one-dimensional sequences $(\alpha_{m,0})_{m=0}^{\infty}$ and $(\alpha_{0,n})_{n=0}^{\infty}$ that are calculated in Subsections 6.1 and 6.2.

7. Implementation

7.1. **Partition of the interval for k.** First, the program partitions the k-axis using the code in Algorithm 3, with numbers κ_0, κ_1, …, κ_m, as a part of precomputation phase. Higher precision results in larger values of m and finer partitions of the k-axis. In our implementation, where the required precision is $\varepsilon = 10^{-7}$, the degree of the polynomials is 8, and $(k,c) \in \left[10^{-6}, 5\right] \times \left[10^{-6}, 0.997\right]$, the number m turns out to be 149. In order to build the partition of the k-axis we need to work with one-dimensional polynomials in which the c coordinate is fixed. We can choose an arbitrary $c_\star$, and in our case we had $c_\star = 0.017$. Then we select the smallest k for which we want to approximate the implied volatility. This is the given left endpoint $k_{\min}$, hence we make the choice $\kappa_0 = k_{\min}$. Then we build the first polynomial approximation P_0 in variable k around $(\kappa_0, c_\star)$ using the PDE method as shown in Subsection 6.2. Then we determine κ_1 as a number for which the error of the approximation $P_0(\kappa_1, c_\star)$ is $\varepsilon/40$. After this we build our second polynomial P_1 around $(\kappa_1, c_\star)$ and obtain the point κ_2 as the one in which the approximation error for $P_1(\kappa_2, c_\star)$ is $\varepsilon/40$. The procedure continues until we reach $k_{\max}$.

Thus, we partitioned the interval $[k_{\min}, k_{\max}]$ with the sequence $\overrightarrow{\kappa} = (\kappa_0, \kappa_1, \ldots, \kappa_m)$ and constructed one-dimensional polynomials $k \mapsto P_i(k, c_\star)$ for $i \in \{1, 2, \ldots, m-1\}$ such that

- $P_i(k, c_\star)$ is a polynomial of degree d;
- $|P_i(k, c_\star) - \Sigma_{BS}(k, c_\star)| \leq \frac{\varepsilon}{40}$ for $k \in [\kappa_i, \kappa_{i+1}]$.

In this step we are calling the bisection method to calculate Σ_{BS}. Bisection method is only used in precomputation phase.

Algorithm 3 Partition of the k axis

Input

ε = desired precision
$[k_{\min}, k_{\max}]$ = domain for the log-moneyness k
d = required degree for polynomials

Output

$\{\kappa_0, \kappa_1, \ldots, \kappa_m\}$ = partition of the segment $[k_{\min}, k_{\max}]$

Code

function KAxisPartition
    $c_\star \leftarrow 0.017$
    $i \leftarrow 0$
    $\kappa_0 \leftarrow k_{\min}$
    while $\kappa_i < k_{\max}$ **do**
        $\overrightarrow{\alpha^K} = \left(\alpha_n^K\right)_{n=0}^d \leftarrow$ Recursion (42) with initial conditions (39), (40), and (41). The center of the expansion k_0 is replaced by κ_i while $\Sigma_0 = \Sigma_{BS}(\kappa_i, c_\star)$ is obtained using bisection method.
        $P(k, c_\star) \leftarrow \sum_{n=0}^d \alpha_n^K \cdot (k - \kappa_i)^n$
        $\kappa_{i+1} \leftarrow$ Maximal k for which $|P(k, c_\star)) - \Sigma_{\mathrm{BS}}(k, c_\star)| < \frac{\varepsilon}{40}$
        $i \leftarrow i + 1$
    $m \leftarrow i$
    return the sequence $\overrightarrow{\kappa} = (\kappa_i)_{i=0}^m$

7.2. **Partitions of the interval for c.** In the previous step we have created the partition $\overrightarrow{\kappa}$. The next task is to create the partitions of the interval for c. The summary is shown in Algorithm 4.

The goal is to iterate over each value $i \in \{0, 1, 2, \ldots, m\}$, fix κ_i, and divide $[c_{\min}, c_{\max}]$ into subintervals on which the polynomials of degree d in c-variable provide an approximation to $\Sigma_{BS}(\kappa_i, \cdot)$ with an error smaller than $\varepsilon/40$. This task is performed by induction. We start with $\sigma_{i0} = \sigma_{\min}$ (in our case we chose $\sigma_{\min} = 10^{-6}$), construct $c_{i0} = C_{BS}(\kappa_i, \sigma_{i0})$ and then find a polynomial Q_{i0} of degree d around (κ_i, c_{i0}) as described in Subsection 6.1. We then start with value $\sigma'_{i1} = \sigma_{i0} + 1.6$, calculate $c'_{i1} = C_{BS}(\kappa_i, \sigma'_{i1})$ and check whether $Q_{i0}(\kappa_i, c'_{i1})$ is within $\varepsilon/40$ from σ'_{i1}. If it is, then we select $c_{i1} = c'_{i1}$, otherwise we repeat the procedure with $\sigma''_{i1} = \sigma_{i0} + 0.8$. Eventually, we arrive to c_{i1} and repeat algorithm to get c_{i2}, c_{i3}, ... until we reach $\sigma_{\max}$.

Observe that the number of cells along the c-axis is not the same for each κ_i, rather it varies significantly and takes values from 119 to 298. When $\varepsilon = 10^{-7}$ the total number of rectangular cells in the partition of D is 29474.

Since this is a part of the algorithm that is executed only once, it is not essential to fully optimize it and perform it in parallel. However, it is possible to perform the treatment of different values of κ_i in parallel. We saw a small gain in performance by doing so.

7.3. **Creation of polynomials for each cell in partition.** For each κ_i and each $c_{i,j}$ we now create a two-dimensional polynomial $P_{i,j}$ around $(\kappa_i, c_{i,j})$ as described in Subsection 6.3 and outlined in Algorithm 5. This task also belongs to the pre-computation phase of the algorithm.

Algorithm 4 Partition of the c axis

Input

ε = desired precision
$\{\kappa_0, \kappa_1, \ldots, \kappa_m\}$ = partition of the segment $[k_{\min}, k_{\max}]$
$[k_{\min}, k_{\max}]$ = domain for the normalized option price c
$[\sigma_{\min}, \sigma_{\max}]$ = domain for integrated volatility
d = required degree for polynomials

Output

$\{c_{i,0}, c_{i,1}, \ldots, c_{i,\max_i}\}$ = partitions of the segment $[c_{\min}, c_{\max}]$ for each $i \in \{0, 1, \ldots, m\}$

Code

function CAxisPartition
  for $i \in \{0, 1, \ldots, m\}$ **do**
    $\sigma_{\text{next}} \leftarrow \sigma_{\min}$, $j \leftarrow 0$, $c_{i,0} \leftarrow C_{BS}(\kappa_i, \sigma)$
    while $c_{i,j} < c_{\max}$ and $\sigma < \sigma_{\max}$ **do**
      $\overrightarrow{\alpha^C} = \left(\alpha^C\right)_{n=0}^{d} \leftarrow$ Recursion (33) where initial conditions are (24), (25), and (26). The quantities Σ_0, k_0, and c_0 are replaced by σ_{next}, κ_i, and $c_{i.j}$, respectively.
      $P(\kappa_i, c) \leftarrow \sum_{m=0}^{d} \alpha_m^C (c - c_{i,j})^m$
      $\sigma_{\text{next}} \leftarrow \sigma + 1.6$, $c_{\text{next}} \leftarrow C_{BS}(k_i, \sigma_{\text{next}})$
      while $|P(\kappa_i, c_{\text{next}}) - \sigma_{\text{next}}| > \frac{\varepsilon}{40}$ **do**
        $\sigma_{\text{next}} \leftarrow \frac{\sigma + \sigma_{\text{next}}}{2}$, $c_{\text{next}} \leftarrow C_{BS}(\kappa_i, \sigma_{\text{next}})$
      $c_{i,j+1} \leftarrow c_{\text{next}}$, $j \leftarrow j + 1$
    $\max_i \leftarrow j$
  return the sequences $\overrightarrow{c_i} = (c_{i,j})_{j=0}^{\max_i}$ for $i \in \{0, 1, \ldots, m\}$

7.4. **Evaluation of implied volatility.** This is the runtime phase of the algorithm. At this point we assume that the partitions and polynomials are created and pre-loaded in the memory. For a given pair (k, c) we first need to find the indices i and j that determine the polynomial $P_{i,j}(k, c)$ to be evaluated.

This task can be done in $O(1)$ time and without any branching. We first locate the index $i \in \{0, 1, 2, \ldots, m-1\}$ such that $k \in (\kappa_i, \kappa_{i+1})$ and for that particular i we locate the index j and the interval $(c_{i,j}, c_{i,j+1})$ to which c belongs.

We explain the procedure to find the index i (same applies to j). Rather than providing a formal description, we proceed with an example. Assume that interval $A = [0, 100)$ is split into three sets

$$A_0 = [0, 2.8), \quad A_1 = [2.8, 33.5), \quad \text{and } A_2 = [33.5, 100].$$

For every $x \in [0, 100)$ we need to find the index $i(x) \in \{0, 1, 2\}$ such that $x \in D_{i(x)}$.

We define the sequence $(I_\ell)_{\ell=0}^{999}$ of 1000 integers in the following way:

$$I_0 = \cdots = I_{27} = 0, \quad I_{28} = \cdots = I_{334} = 1, \quad I_{335} = \cdots = I_{999} = 2.$$

Given $x \in [0, 100)$, we first calculate the integral part $\lfloor 10x \rfloor$ of $10x$. The value $i(x)$ is defined as $i(x) = I_{\lfloor 10x \rfloor}$.

Algorithm 5 Creation of the polynomial

Input

1: (c_0, k_0) = the center of the expansion
2: d = required degree

Output

1: $(\alpha_{m,n})_{m,n=0}^{d}$ = coefficients of the polynomial

Code

1: **function** CREATEPOLYNOMIAL
2: $\Sigma_0 \leftarrow \Sigma_{BS}(k_0, c_0)$. The calculation is performed using bisection method.
3: $\overrightarrow{\alpha^K} = \left(\alpha_n^K\right)_{n=0}^{d} \leftarrow$ Recursion (42) with initial conditions (39), (40), and (41).
4: $\overrightarrow{\alpha^C} = \left(\alpha^C\right)_{n=0}^{d} \leftarrow$ Recursion (33) where initial conditions are (24), (25), and (26).
5: **for** $i \in \{0, 1, \ldots, d\}$ **do**
6: $\alpha_{i,0} \leftarrow \alpha_i^C$
7: $\alpha_{0,i} \leftarrow \alpha_i^K$
8: $(\alpha_{u,v})_{u,v=1}^{d} \leftarrow$ Recursion (56).
9: **return** the sequence $(\alpha_{m,n})_{m,n=0}^{d}$

7.5. **Approximation to volatility surface.** For our chosen level of precision $\varepsilon = 10^{-7}$ the actual approximation surface is quite smooth and virtually indistinguishable from the true $\Sigma_{BS}(k, c)$. In order to demonstrate its rough piecewise structure, we implemented our algorithm with precision $\varepsilon = 10^{-1}$ and polynomials of degree 5. Figure 2 shows five cross sections corresponding to the values $k \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Notice that different values for k generate different interval partitions on the c-axis, as already depicted in Figure 1.

8. Analysis of performance

The precomputation phase of the algorithm consists of creating the partition and polynomials for a given precision ε and domain D for pairs (k, c). This step has to be performed only once and the results can be stored for future use. When the choices are $\varepsilon = 10^{-7}$ and $D = \left[10^{-6}, 5\right] \times \left[10^{-6}, 0.997\right]$ it takes less than a minute to create the partition and polynomials. Once the set-up phase is completed, the evaluation of implied volatility is faster than when using other methods, as it can be seen in the table below.

Method	Time (in seconds)	Domain	Error
Jäckel [23]	0.13498	D	$5.35 \cdot 10^{-13}$
Li SOR [33]	0.16054	D	$3.2 \cdot 10^{-6}$
PDE on CPU	0.05276	D	10^{-7}
PDE on GPU	0.00925	D	10^{-7}

The tests consist of one million randomly generated option prices and were run on Amazon Web Services using an instance of the type p3.2xlarge. The instance is powered by an Intel Xeon E5-2686 v4 (Broadwell) processor with 8 virtual cores and has an NVIDIA Tesla V100 GPU with 5120 CUDA Cores. The first three rows

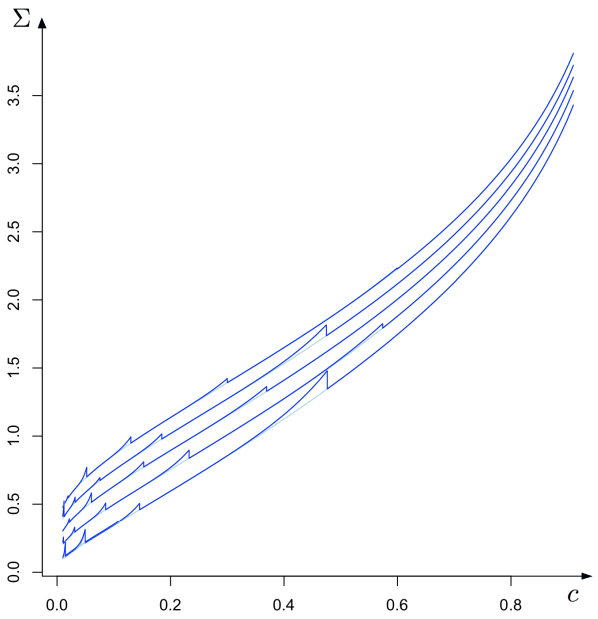

FIGURE 2. Graphs of $c \mapsto \Sigma(k, c)$ for $k \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and their approximations with tolerance 10^{-1}.

of the table correspond to the programs that were executed on CPU. All 8 cores were utilized using the openMP framework. As the table shows, the PDE method implemented on the graphic card with openCL framework further speeds up our execution by a factor of 5.

Keep in mind that our power series expansions are divergent for $k = 0$. Thus, the implied volatilities cannot be calculated directly for ATM options. However, our algorithm deals with this case by simply replacing $k = 0$ with $k = 1.01 \cdot 10^{-7}$. The obtained implied volatility is still within $\varepsilon = 10^{-7}$ from its true value.

There are alternative methods available for the ATM case that can be combined with our PDE method while still maintaining a small overall absolute error. One such method that could show promising performance is to use the expansion of Grunspan (Proposition 6 from [19]). Another approach is discussed in [38]. A simple closed-form approximation function $\Sigma_{SR}(k, c)$ for $\Sigma_{BS}(k, c)$ was derived in [49], based on the Pólya approximation function $\tilde{N}(\cdot)$ of the normal CDF $N(\cdot)$. The function $\Sigma_{SR}(k, c)$ is cheap to compute, and moreover it was shown in [16] that it is a lower bound for $\Sigma_{BS}(k, c)$ for most pairs (k, c) that are relevant in practice. The approximation Σ_{SR} is actually very accurate for small values of c which makes it convenient to combine with our PDE method whose cells are very dense and numerous for such c. More specifically, we tried using Σ_{SR} instead of our PDE method when $k < 0.0016$ and $c < 0.063$. The approximation happens to be 20% faster and the space for storing the partition and polynomials is reduced by

the factor of 3. However, the error of such approximation turns out to be $5 \cdot 10^{-5}$. Tests and results reported in the table above do not use any of these methods.

Our method also yields an improved repricing error, which was used as one of the benchmarks in [45] and is defined as

$$\varepsilon_r \quad = \quad \max_{(k,c)} \left| C_{BS}\left(k, \Sigma(k,c)\right) - c \right|,$$

where C_{BS} is defined in (4). The corresponding repricing error for the PDE method with parameters given above is 10^{-6}. It is better than 10^{-4} error achieved by [32] and 10^{-5} error achieved by [45]. Note that the repricing error of 10^{-4} may be substantial once it is multiplied by an underlying of high magnitude.

The algorithm was also tested on a data sample of options provided to us by Wharton Research Data Services. We chose February 18, 2016 as an arbitrary day in history for which option data was available. We pruned the data for meaningful quotes that satisfy put-call parity, non-zero bid-ask spreads and a reasonable trading volume. The corresponding re-scaled values k and c for all of the non-ATM options were within the domain $D = \left[10^{-6}, 5\right] \times \left[10^{-6}, 0.997\right]$, hence the PDE method achieved the precision 10^{-7}. In fact the smallest c that occurred in the dataset was $2 \cdot 10^{-5}$. On this real-world data our PDE method was about three times faster than other methods. As in [45] we did not observe any bid-ask range violations once our implied volatility approximation is plugged into the Black-Scholes call option price formula.

9. Implied volatility in Bachelier model

9.1. **Introduction.** Bachelier [1] created a model for prices of securities decades before Black-Scholes. The underlying dynamics follows arithmetic Brownian motion with volatility σ_{Bac}.

$$dS_t \quad = \quad \sigma_{\text{Bac}}\, dW_t. \tag{57}$$

The price of the call option with strike K is given by

$$C \quad = \quad (S_0 - K)\, N(d) + \sigma\sqrt{T} \cdot N'(d) \quad \text{where} \tag{58}$$

$$d \quad = \quad \frac{S_0 - K}{\sigma_{\text{Bac}}\sqrt{T}}. \tag{59}$$

We only consider call options here, as put-call parity for Bachelier model makes it easy to relate the two. For the sake of simplicity we assume that the stock pays no dividends and that the interest rate is zero. The general case can be handled in a very similar way.

The volatility parameter σ_{Bac} is often called *Bachelier volatility*, *normal volatility*, or *basis point volatility*. One characteristic of this model is that the underlying asset can get a negative value. Nevertheless, it is still appropriate for short maturities and interest rate derivative markets [28]. An interesting comparison between Bachelier and Black-Scholes model is done by Schachermayer and Teichmann [46] who showed that for short maturities the former offers a good approximation for the latter. Choi, Kim, and Kwak [9] nicely summarized several quantities that are commonly expressed in terms of the Bachelier implied volatility: spread options, "break-even move" of delta-hedge portfolio, and swaption prices where skewness gets reduced when the Bachelier volatility is used. The Bachelier implied volatility is related to the incomplete gamma function, as observed by Grunspan [20].

Recently Le Floc'h [28] and Jäckel [24] provide almost machine precision approximation for the Bachelier implied volatility based on the rational approximation from [9].

We will illustrate how the idea of our PDE method can be used to obtain very precise approximations to the Bachelier implied volatility. To reduce the problem to one dimension, we divide both sides of (58) by $S_0 - K$. Substituting $c = \frac{C}{S_0 - K}$ gives us the equation $c = C_{\text{Bac}}(d)$, where[7]

$$C_{\text{Bac}}(d) \quad = \quad N(d) + \frac{N'(d)}{d}. \tag{60}$$

Therefore we can formulate the problem of finding the implied volatility in Bachelier model as

$$\text{For given } c \text{ find a positive real number } d \text{ that solves } C_{\text{Bac}}(d) = c. \tag{61}$$

Once d is found, then σ_{Bac} can be recovered from (59).

The problem (61) is equivalent to finding the function $\psi(x)$ such that

$$x \quad = \quad N\left(\psi(x)\right) + \frac{N'\left(\psi(x)\right)}{\psi(x)}. \tag{62}$$

Notice that $x > 1$ and that the right-hand side of (62), as a function of ψ, is decreasing and converges to 1 as $\psi \to +\infty$. The implicit function ψ depends on only one variable, see Figure 3. Recall that in Black-Scholes model, the integrated volatility Σ was a function of two variables k and c. Therefore instead of constructing PDEs we will be able to create an approximation using just ODEs.

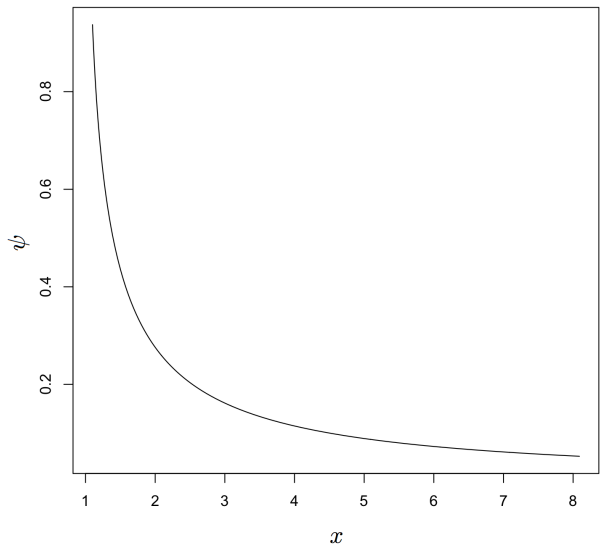

FIGURE 3. Graph of function $\psi(x)$ for $1 < x \leq 8$

[7]From this point on we will assume that $d > 0$ which corresponds to ITM call options. The OTM case is treated by "in-out-duality" [33] when one replaces d by $-d$ in (60) and instead of $c = \frac{C}{S_0 - K}$ defines $c = \frac{C}{K - S_0} + 1$.

It is straightforward to derive the following identity from (14)

$$\left[N'\left(\psi(x)\right)\right]' \quad = \quad -N'\left(\psi(x)\right)\psi'(x)\psi(x). \tag{63}$$

Using (63) and taking derivatives of both sides of (62) implies

$$1 \quad = \quad N'\left(\psi(x)\right)\psi'(x) + \frac{-N'\left(\psi(x)\right)\psi'(x)\psi^2(x) - N'\left(\psi(x)\right)\psi'(x)}{\psi^2(x)}. \tag{64}$$

The last equation is equivalent to

$$-1 \quad = \quad \frac{N'\left(\psi(x)\right)\psi'(x)}{\psi^2(x)}. \tag{65}$$

Taking derivative of both sides of (65) gives us

$$\begin{aligned} 0 \quad = \quad & \frac{1}{\psi^4(x)} \cdot \left\{ \left[-N'\left(\psi(x)\right)\psi(x)\left(\psi'(x)\right)^2 + N'\left(\psi(x)\right)\psi''(x) \right] \cdot \left(\psi(x)\right)^2 \right. \\ & \left. -2N'\left(\psi(x)\right)\psi(x)\left(\psi'(x)\right)^2 \right\}. \end{aligned}$$

Multiplying both sides by $\psi^3(x)$ and dividing by $N'\left(\psi(x)\right)$ gives us

$$\left(2+\psi^2(x)\right)\cdot\left(\psi'(x)\right)^2 \quad = \quad \psi''(x)\cdot\psi(x). \tag{66}$$

Notice that equations (65) and (66) allow us to immediately conclude that ψ is a decreasing and convex function.

9.2. **Power series representation of ψ.** Assume that x_0 and ψ_0 are fixed and that they satisfy $\psi_0 = \psi\left(x_0\right)$.[8] Let us denote by $\left(\alpha_k\right)_{k=0}^{\infty}$ the real numbers such that

$$\psi(x) \quad = \quad \sum_{k=0}^{\infty} \alpha_k \left(x - x_0\right)^k. \tag{67}$$

Then we have $\alpha_0 = \psi_0$. We place ψ_0 instead of $\psi(x)$ in (65), solve for $\psi'(x)$ and call α_1 the obtained solution. The remaining terms of the sequence $\left(\alpha_k\right)_{k=0}^{\infty}$ are determined recursively.

Let us denote by $\mathcal{P}_m$ the set of all pairs of non-negative integers that sum up to m. In other words

$$\mathcal{P}_m \quad = \quad \left\{(p_1,p_2)\in\mathbb{N}_0^2 : p_1+p_2=m\right\}. \tag{68}$$

Define the sequences $\left(\beta_k\right)_{k=0}^{\infty}$ and $\left(\gamma_k\right)_{k=0}^{\infty}$ as

$$\beta_m \quad = \quad \begin{cases} \sum\limits_{\mathcal{P}_m} \alpha_{p_1}\alpha_{p_2}, & \text{if } m \geq 1, \\ \alpha_0^2 + 2, & \text{if } m = 0; \end{cases} \tag{69}$$

$$\gamma_m \quad = \quad \sum_{\mathcal{P}_m} (p_1+1)(p_2+1)\alpha_{p_1+1}\alpha_{p_2+1}. \tag{70}$$

[8]One way to get this pair (x_0,ψ_0) is to start from x_0 and use bisection method to solve $x_0 = C_{\text{Bac}}\left(\psi_0\right)$ for ψ_0. Alternatively, one can start from ψ_0 and evaluate x_0 explicitly as $C_{\text{Bac}}\left(\psi_0\right)$.

From equation (66) we obtain the recursion for the sequence $(\alpha_k)_{k=0}^{\infty}$:

$$\alpha_{m+2} = \frac{1}{(m+2)(m+1)\alpha_0}\left(\sum_{j=0}^{m}\beta_j\gamma_{m-j} - \sum_{p=0}^{m-1}(p+1)(p+2)\alpha_{p+2}\alpha_{m-p}\right). \tag{71}$$

9.3. **Simulations.** We analyze the performance of the algorithm when the desired error is $\varepsilon = 10^{-6}$ with polynomials of degree 10. When restricted to $x \in [1.04, 8]$, we need 16 polynomials to achieve the error 10^{-6}, each of them providing an approximation on a sub-interval of $[1.04, 8]$. The endpoints of the subintervals are

1.037649, 1.050247, 1.067051, 1.089598, 1.120133, 1.162058, 1.22078, 1.30545, 1.432885, 1.637783, 2.005211, 2.317769, 2.814489, 3.71644, 4.509353, 5.834184, 8.488817.

Observe that the intervals of the partition are denser near $x = 1$ which is reminiscent of the behavior of the two-dimensional partition of the (k, c) plane in Figure 1. The error of the above approximation is shown in Figure 4.

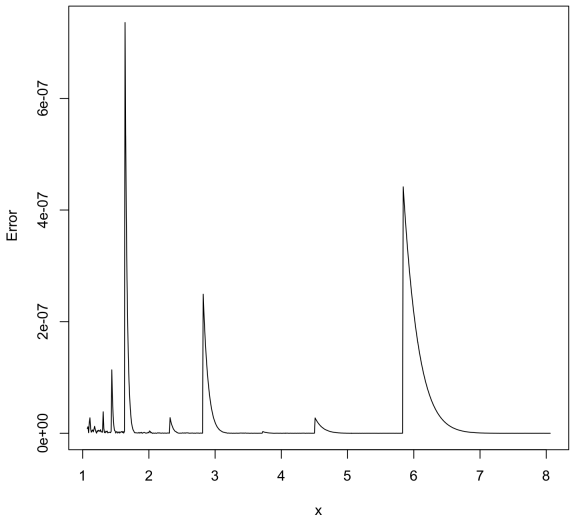

FIGURE 4. Error for Bachelier approximation with polynomials of degree 10 with 16 intervals.

10. Further directions

10.1. **Drawbacks of the PDE method.** Our choices for parameters $c_\star$, ε, threshold $\varepsilon/40$ (in Algorithms 3 and 4) and the degree of polynomials used in approximations were obtained by trial and error in our experiments. We are well aware that

there is a possibility that more optimal choices could have been made. However, that will likely require more time and space complexity.

Our implementation uses polynomials of degree 8 and achieves the 10^{-7} precision. The polynomials of lower degree are easier to handle by the graphic card, but they require more partition cells. This results in bigger data structures that have to be stored in memory. The polynomials of higher degree require more computation but smaller files. We have observed that degree 8 polynomials offer the best compromise and result in fastest execution. In the near future memory capacities of graphic cards may allow more efficient implementations and better precision.

The power series (19) is divergent at $k = 0$ which corresponds to ATM options. We overcame this difficulty by replacing $k = 0$ with $k = 1.01 \cdot 10^{-7}$ and maintained the same absolute error. It remains to be seen whether there are superior ways of handling this case.

The *existence* of the solution to the system of partial differential equations (1), (2), and (3) is established in Theorem 1. This solution is the implied volatility Σ_{BS} and our numerical simulations show that low degree truncations of the power series (19) are precise approximations of Σ_{BS}. The *uniqueness* of the solutions to the system of partial differential equations (1), (2), and (3) remains an open problem. If for a fixed pair (k_0, c_0) the power series (19) is convergent, then it is the unique solution to the system. The question of uniqueness is thus equivalent to the convergence of the power series (19) in a neighborhood of (k_0, c_0). We conjecture that the system of partial differential equations (1), (2), and (3) has a unique solution.

Conjecture 1. *For every pair* (k_0, c_0) *and every* $\Sigma_0 > 0$ *such that* $k_0 \neq 0$ *and* $c_0 > 0$*, the system of partial differential equations* (1), (2)*, and* (3) *has a unique solution* Σ *in a neighborhood of* (k_0, c_0) *that satisfies* $\Sigma(k_0, c_0) = \Sigma_0$.

The recursive definition of the coefficients $\alpha_{m,n}$ makes it difficult to study the convergence of the series. One possible approach is to adapt the Lagrange inversion methodology Xia and Cui used in [54] to obtain the one-dimensional power series representation equivalent to (27) for fixed k.

10.2. **Applications to other problems.** Our PDE method can be used to approximate the Margrabe implied correlation as suggested in [33]. If two assets have dynamics that is governed with ρ-correlated Wiener processes W_1 and W_2 then the Margrabe formula gives the time 0 price C_0 of the European option to exchange one stock for the other at time T as follows

$$\begin{aligned} C_0 &= S_1(0)e^{-q_1 T}N(d_+) - S_2(0)e^{-q_2 T}N(d_-), \quad \text{where} \qquad (72)\\ d_\pm &= \frac{\log \frac{S_1(0)e^{-q_1 T}}{S_2(0)e^{-q_2 T}} \pm \frac{\sigma^2 T}{2}}{\sigma\sqrt{T}}, \quad \text{and} \\ \sigma_M &= \sqrt{\sigma_{BS,1}^2 + \sigma_{BS,2}^2 - 2\rho\sigma_{BS,1}\sigma_{BS,2}}. \qquad (73) \end{aligned}$$

The quantities q_1 and q_2 are the dividend rates for the underlying assets while $\sigma_{BS,1}$ and $\sigma_{BS,2}$ are their volatilities. After the substitution

$$k = \log \frac{S_2 e^{-q_2 T}}{S_1 e^{-q_1 T}}, \quad c = \frac{C_0}{S_1 e^{-q_1 T}}, \quad \sigma = \sigma_M\sqrt{T},$$

the formula (72) becomes

$$c \quad = \quad C_{BS}(k, \sigma), \tag{74}$$

where the function C_{BS} is defined in (4).

In practice, one can use the PDE method to first find $\sigma_{BS,1}$ and $\sigma_{BS,2}$. Then we use the same method to invert the equation (**??**) for σ. Finally the implied correlation ρ can be obtained from (73).

Similarly, one can use our PDE method to obtain the critical stock price in a compound option as shown in [33].

10.3. **Extensions of theoretical results.** It is expected that Theorem 1 can be used to obtain further theoretical results about implied volatility. For example, one such result is

Theorem 2. *The implied volatility $\Sigma(k, c)$ satisfies the following equation*

$$\frac{k}{\Sigma^2} - \frac{1}{2} = \frac{\partial}{\partial c}\left(\frac{\Sigma_k}{\Sigma_c}\right). \tag{75}$$

Proof. Using elementary algebraic transformations we can re-write equation (3) as

$$\Sigma^3 \Sigma_{kc} \quad = \quad \Sigma_c \Sigma_k \left(\frac{\Sigma^4}{4} - k^2\right) - \Sigma \Sigma_c \left(\frac{\Sigma^2}{2} - k\right). \tag{76}$$

Using the relation (1) we obtain that the first term on the right-hand side of (76) satisfies

$$\Sigma_c \Sigma_k \left(\frac{\Sigma^4}{4} - k^2\right) = \Sigma^3 \Sigma_{cc} \frac{\Sigma_k}{\Sigma_c}.$$

Multiplying both sides of (76) by $\frac{1}{\Sigma_c \Sigma^3}$ and using the previous equation gives us

$$\frac{\Sigma_{kc}\Sigma_c - \Sigma_{cc}\Sigma_k}{\Sigma_c^2} = \frac{k}{\Sigma^2} - \frac{1}{2}. \tag{77}$$

Applying the quotient rule to the left-hand side of (77) implies (75). □

An easy consequence of Theorem 2 is the following curious convexity result about Black-Scholes implied volatility.

Corollary 1. *For every fixed k the function $f(c) = \frac{\Sigma_k(k,c)}{\Sigma_c(k,c)}$ is concave.*

Proof. Differentiating both sides of (75) with respect to c implies

$$f''(c) = -\frac{2k\Sigma\Sigma_c}{\Sigma^4}.$$

Since Σ and Σ_c are positive, we obtain that $f''(c)$ is negative, which implies the required result. □

Tehranchi in [52] established similar convexity results for the integrated volatility Σ (see Corollary 1.2).

We also note that Theorem 1 can be concisely stated in terms of the Hessian $D^2\Sigma$ as follows.

Theorem 3. *The Hessian $D^2\Sigma$ of the function Σ satisfies the following partial differential equation*

$$\Sigma^3 D^2\Sigma \;=\; \begin{bmatrix} \mathcal{A}(\Sigma)+\mathcal{B}(\Sigma) & 0 \\ 0 & \mathcal{C}(\Sigma) \end{bmatrix} \cdot \begin{bmatrix} \mathcal{A}(\Sigma)-\mathcal{B}(\Sigma) & \mathcal{C}(\Sigma) \\ \mathcal{A}(\Sigma)+\mathcal{B}(\Sigma) & \mathcal{D}(\Sigma) \end{bmatrix},$$

where $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ $: C^1\left(\mathbb{R}^2\right) \to C\left(\mathbb{R}^2\right)$ are defined as

$$\mathcal{A}(\Sigma) = \frac{1}{2}\Sigma^2\frac{\partial}{\partial k}\Sigma, \qquad \mathcal{B}(\Sigma) = k\frac{\partial \Sigma}{\partial k} - \Sigma,$$
$$\mathcal{C}(\Sigma) = \left(\frac{\Sigma^2}{2} - k\right)\frac{\partial}{\partial c}\Sigma, \qquad \mathcal{D}(\Sigma) = \left(\frac{\Sigma^2}{2} + k\right)\frac{\partial}{\partial c}\Sigma.$$

It can be easily verified from here that the determinant of the Hessian satisfies

$$\det\left(D^2\Sigma\right) \;=\; \frac{1}{2}\Sigma_c^2 \cdot \frac{\partial}{\partial k}\left(\left(-\frac{k}{\Sigma} + \frac{\Sigma}{2}\right)^2\right)$$

which may be useful for gradient-based optimization procedures.

The system of partial differential equations (1), (2), and (3) might be numerically solvable using a finite difference method. It remains to be seen whether this approach would yield effective algorithms that provide accuracy and speed desired in practice.

11. Concluding remarks

We have proved that the integrated volatility satisfies a system of nonlinear second order partial differential equations. The obtained equations are used to construct a recursive algorithm for determining the power series approximation of the integrated volatility. We demonstrated how our approximation can be used to obtain faster algorithms for implied volatility and presented the detailed results of the comparison we made between this new approximation and existing methods. The gain in speed is substantial, especially when the algorithm is parallelized. We also demonstrated that the partial differential equations offer additional insights into the behavior of implied volatility.

Acknowledgements

This work was supported by the Weissman School of Arts and Sciences at Baruch College. The authors thank their colleagues Beat Ammon, Jim Gatheral, Andrew Lesniewski, Yike Lu, Elena Kosygina, Avi Palley, Alex Tartakovsky, and anonymous referees for their comments and suggestions.

References

[1] L. Bachelier. Théorie de la spéculation. *Annales Scientifiques de L'Ecole Normale Supérieure*, 17, pp. 21–88, 1900.

[2] J. Baumeister. Inverse problems in finance. in: *Recent Developments in Computational Finance: Foundations, Algorithms and Applications*, T. Gerstner, P. Kloeden (Eds.), World Scientific Publishing, Volume 14, *Interdisciplinary Mathematical Sciences*, Chapter 3, pp. 81–158, 2013.

[3] H. Berestycki, J. Busca, and I. Florent. Computing the implied volatility in stochastic volatility models. *Comm. Pure Appl. Math.*, 57(10), pp. 1352–1373, 2004.

[4] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81, pp. 637–654, 1973.
[5] M. Brenner and M. Subrahmanyam. A simple formula to compute the implied standard deviation. *Financial Analyst Journal*, 44, pp. 80–83, 1988.
[6] M. Brenner and M. Subrahmanyam. A simple approach to option valuation and hedging in the Black-Scholes model. *Financial Analyst Journal*, 50, pp. 25–28, 1994.
[7] D. M. Chance. A generalized simple formula to compute the implied volatility. *The Financial Review*, 31, pp. 859–867, 1996.
[8] D. Chance, T. A. Hanson, W. Li, and J. Muthuswamy. A bias in the volatility smile. *Review of Derivatives Research*, 20 (1), pp. 47–90, 2017.
[9] J. Choi, K. Kim, and M. Kwak. Numerical approximation of the implied volatility under arithmetic Brownian motion. *Applied Mathematical Finance*, 16 (3), pp. 261–268, 2009.
[10] C. J. Corrado and T. W. Miller Jr. A note on a simple, accurate formula to compute implied standard deviations. *Journal of Banking and Finance*, 20, pp. 595–603, 1996.
[11] M. R. Fengler. Option data and modeling BSM implied volatility. in: *Handbook of Computational Finance*, pp. 117–142. Springer Berlin Heidelberg, 2012.
[12] M. Forde and A. Jacquier. Small-time asymptotics for implied volatility under the Heston model. *International Journal of Theoretical and Applied Finance* 12 (6), pp. 861–876, 2009.
[13] G. Frobenius. Ueber die Integration der linearen Differentialgleichungen durch Reihen. J. reine angew. Math., 76, pp. 214–235, 1873.
[14] J. Gatheral. *The Volatility Surface: A Practitioner's Guide*. John Wiley and Sons, Hoboken, NJ, 2006.
[15] J. Gatheral, E. P. Hsu, P. Laurence, C. Ouyang, and T.-H. Wang. Asymptotics of implied volatility in local volatility models. *Mathematical Finance* 22 (4), pp. 591–620, 2012.
[16] J. Gatheral, I. Matić, D. Stefanica, and R. Radoičić. Tighter bounds for implied volatility. *International Journal of Theoretical and Applied Finance* 20 (5), pp. 1–14, 2017.
[17] S. Gerhold. Can there be an explicit formula for implied volatility? *Appl. Math. E-Notes*, 13, pp. 17–24, 2013.
[18] K. Glau, P. Herold, D. Madan, C. Pötz. The Chebyshev method for the implied volatility. *Journal of Computational Finance*, to appear, *arXiv:1710.01797*, 2017.
[19] C. Grunspan. Asymptotic Expansions of the Lognormal Implied Volatility: A Model Free Approach. *arXiv:1112.1652*, 2011.
[20] C. Grunspan. A Note on the Equivalence between the Normal and the Lognormal Implied Volatility: A Model Free Approach. *arXiv:1112.1782*, 2011.
[21] M. Hairer. A theory of regularity structures. Inventiones mathematicae, 198(2), pp. 269–504, 2014.
[22] P. Jäckel. By implication. *Wilmott*, November, pp. 60–66, 2006.
[23] P. Jäckel. Let's be rational. *Wilmott*, January, pp. 40–53, 2015.
[24] P. Jäckel. Implied normal volatility. *Wilmott*, January, pp. 52–54, 2017.
[25] A. Jacquier, M. Keller–Ressel, and A. Mijatović. Large deviations and stochastic volatility with jumps: asymptotic implied volatility for affine models. *Stochastics* 85 (2), pp. 321–345, 2013.
[26] A. Jacquier and M. Lorig. From characteristic functions to implied volatility expansions. *Advances in Applied Probability* 47 (3), pp. 837–857, 2015.
[27] H. A. Latané and R. J. Rendleman. Standard deviation of stock price ratios implied by option premia. *Journal of Finance*, 31, pp. 369–382, 1976.
[28] F. Le Floc'h. Fast and Accurate Analytic Basis Point Volatility (April 10, 2016). Available at SSRN: https://ssrn.com/abstract=2420757
[29] F. Le Floc'h. Implied volatility from Black-Scholes price. (2017) Available at *http://chasethedevil.github.io/post/implied-volatility-from-black-scholes-price/*
[30] R. Lee. The moment formula for implied volatility at extreme strikes. *Mathematical Finance*, 14 (3), pp. 469–480, 2004.
[31] S. Li. A new formula on computing the implied volatility. *Applied Mathematics and Computation*, 170, pp. 611–625, 2005.
[32] M. Li. Approximate inversion of the Black-Scholes formula using rational functions. *European Journal of Operational Research*, 185 (8), pp. 743–759, 2008.
[33] M. Li and K. Lee. An adaptive successive over-relaxation method for computing the Black-Scholes implied volatility. *Quantitative Finance*, 11 (8), pp. 1245–1269, 2011.

[34] M. Lorig, S. Pagliarani, and A. Pascucci. A Taylor series approach to pricing and implied volatility for local-stochastic volatility models. *Risk*, 17 (2), pp. 1–17, 2014.
[35] D. B. Madan. Adapted hedging. *Annals of Finance*, 12 (3-4), 305–334, 2016.
[36] D. B. Madan, R. H. Smith, and K. Wang. Laplacian risk management. *Finance Research Letters*, 22, pp. 202–210, 2017.
[37] S. Manaster and G. Koehler. The calculation of implied variances from the Black-Scholes model. *The Journal of Finance*, 38, pp. 227–230, 1982.
[38] I. Matić, R. Radoičić, and D. Stefanica. Pólya-based approximation for the ATM-forward implied volatility. *International Journal of Financial Engineering*, 4 (2-3), 2017.
[39] I. Matić, R. Radoičić, and D. Stefanica. A PDE method for estimation of implied volatility, source code. *open source, available at github.com/maticivan/PDE-method-for-implied-volatility*
[40] E. A. Nurminskii, A. A. Buryi. Parker-Sochacki method for solving systems of ordinary differential equations using graphics processors. *Numerical Analysis and Applications*, 4:223, 2011.
[41] G. Orlando and G. Taglialatela. A review on implied volatility computation. *Journal of Computational and Applied Mathematics*, 320, 202–220, 2017.
[42] G. E. Parker and J. S. Sochacki. Implementing the Picard iteration. *Neural, Parallel Sci. Comput.*, 4(1), pp. 97–112, 1996.
[43] M. Pistorius and J. Stolte. Fast computation of vanilla prices in time-changed models and implied volatilities using rational approximations. *International Journal of Theoretical and Applied Finance* 15 (4), 1–34, 2012.
[44] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Second Edition. Cambridge University Press, New York, NY, 2002.
[45] O. Salazar Celis. A parametrized barycentric approximation for inverse problems with application to the BlackScholes formula. *IMA Journal of Numerical Analysis*, drx020, 2017.
[46] W. Schachermayer and J. Teichmann. How close are the option pricing formulas of Bachelier and Black-Merton-Scholes? Mathematical Finance 18 (1), 155–170, 2008.
[47] K. Spanderen. Adaptive SOR method for implied volatility calculation. *https://hpcquantlib.wordpress.com/*, 2017.
[48] D. Stefanica and R. Radoičić. A sharp approximation for ATM-forward option prices and implied volatilities. *International Journal of Financial Engineering*, 3(1), 2016.
[49] D. Stefanica and R. Radoičić. An explicit implied volatility formula. *International Journal of Theoretical and Applied Finance*, 20(7), 2017.
[50] G. Steinbrecher and W. T. Shaw. Quantile mechanics. *European Journal of Applied Mathematics*, 19 (2), pp 87–112, 2008.
[51] M. R. Tehranchi. Uniform bounds for Black–Scholes implied volatility. *SIAM Journal on Financial Mathematics*, 7(1), pp. 893–916, 2016.
[52] M. R. Tehranchi. A Black–Scholes inequality: applications and generalisations. *arXiv:1701.03897*, 2017.
[53] A. Townsend and L. N. Trefethen. An extension of chebfun to two dimensions. *SIAM Journal on Scientific Computing*, 35(6), pp. 495-518, 2013.
[54] Y. Xia and Z. Cui. An Exact and Explicit Implied Volatility Inversion Formula. *International Journal of Financial Engineering*, 5(3), 29 pages, 2018.

Baruch College, City University of New York
E-mail address: ivan.matic@baruch.cuny.edu

Baruch College, City University of New York
E-mail address: rados.radoicic@baruch.cuny.edu

Baruch College, City University of New York
E-mail address: dan.stefanica@baruch.cuny.edu