

# *Path Integral Methods for Parabolic Partial Differential Equations*

*With Examples from Computational Finance*

## **Andrew Lyasoff**

By using specific examples—mostly from computational finance—this paper shows that *Mathematica* is capable of transforming Feynman-type integrals on the pathspace into powerful numerical procedures for solving general partial differential equations (PDEs) of parabolic type, with boundary conditions prescribed at some fixed or free time boundary. Compared to the widely used finite difference methods, such procedures are more universal, are considerably easier to implement, and do not require any manipulation of the PDE. An economist or a financial engineer who works with very general models for asset values, including the so-called stochastic volatility models, will find that this method is just as easy to work with as the classical Black–Scholes model, in which asset values can follow only a geometric Brownian motion process.

## ■ 1. Introduction

Parabolic partial differential equations (PDEs), of which the standard heat equation is a special case, are instrumental for many applications in physics, engineering, economics, finance, weather forecasting, and many other fields. In practice, the solution to most parabolic PDEs can be computed only approximately by way of some special numerical procedure. Currently, in most if not all practical implementations, this procedure happens to be some variation of the well-known *finite difference method*.

This method involves tools and terminology that are familiar to most specialists in numerical methods. Its underlying principle is relatively easy to understand: it comes down to solving sequentially a very large number of linear systems.

Unfortunately, in most applications the size of each system is overwhelming and the practical implementation of the method is not straightforward, especially in the case of non-constant coefficients.

It has been well known for nearly as long as the finite difference method has been in existence that the solution of the heat equation can be expressed as a path integral (also known as a Feynman integral). For a mathematician, this is nothing but an expected value relative to Wiener's probability distribution on the space of sample paths. This construction also admits a generalization: the solution of a fairly general parabolic PDE can be expressed as an expected value (alias: a Feynman integral) relative to a special probability distribution on the pathspace (alias: the Wiener measure). Although possible in principle, the numerical calculation of path integrals has always been seen as a formidable task and a rather inefficient way to produce a solution to a parabolic PDE. As the examples included in this article demonstrate, not only is this no longer the case, but, with the advent of *Mathematica* 4, the solution to a rather general parabolic PDE can be produced by merely expressing the associated path integral in *Mathematica*.

The method developed in this paper is a good illustration of the way in which significant new developments in the common man's computing tools transform the type of mathematics that is commonly perceived as interesting and important. We will show with concrete examples (see Section 4 below) that computing the price of an European call option—a classical problem in computational finance—in terms of path integrals is just as easy, just as accurate, and virtually just as fast as calculating the price directly by using the Black–Scholes formula. In other words, for all practical purposes, the path integral representation of the solution of the Black–Scholes PDE in *Mathematica* is just as good as a closed form solution. Unlike the true closed form solution, however, the path integral representation can be used with any reasonably behaved payoff. Of course, this approach would have been unthinkable when Black and Scholes developed their method [1]. Their calculation was interesting and profoundly important, although the really important discovery that they made was the principle that led to the Black–Scholes PDE—not the Black–Scholes formula, as is commonly believed.

It is remarkable how *Mathematica*'s ability to combine advanced symbolic computing with advanced numerical procedures can turn a fairly abstract mathematical concept (in this case, integration on the pathspace) into a rather practical computing tool. This brings a method to consumers of computing technology that is both universal and “smart,” in the sense that it is capable of taking advantage of the structure of the PDE. For example, the method will need fewer iterations if the coefficients are close to linear, and can work on a very coarse grid in the state space if the solution is reasonably close to a polynomial. Above all, its implementation is extremely straightforward, involves very little programming, and does not require any knowledge of sophisticated numerical methods, which *Mathematica* uses implicitly. For a financial engineer, this means that pricing an option with any reasonably-behaved payoff function on an asset that follows a rather general time-homogeneous Markov process, including options that allow early exercise, would be just as easy as pricing a vanilla European call or put option on an asset

that follows a standard geometric Brownian motion process. Such technology can have a considerable impact on the financial industry. Indeed, it has long been recognized that actual stock prices exhibit self-regulatory patterns that are inconsistent with the properties of the geometric Brownian motion. Yet the Black–Scholes model remains immensely popular mainly because, with the current level of widely available computing technology, any other model would be virtually intractable for an average practitioner.

Finding the most efficient way to present the computational aspects of the path integral method and its implementation in *Mathematica* is a bit of a challenge. Indeed, on the one hand, specialists in (and consumers of) numerical methods for PDEs do not usually speak the language of path integrals. On the other hand, specialists in mathematical physics and probability, who do speak the language of path integrals, are not usually interested in numerical methods and computer programming. This article grew out of the desire to meet this challenge. In spite of its use of high-end mathematical jargon, it is mainly about *Mathematica* and the impact that *Mathematica* can make not only on the way in which important computational problems are being solved, but also on our understanding of which mathematical methods are “applied” and which are “theoretical.”

**Note about the notation:** Following the common convention in *Mathematica*, in all equations and algebraic expressions, arguments of functions are wrapped with square brackets and parenthesis are used only for grouping. Double-struck square brackets are used to denote intervals: open intervals are denoted as  $]a, b[$  and closed intervals are denoted as  $[[a, b]]$ .

For the purpose of illustration, all time-consuming operations in this notebook are followed by `//Timing`. The CPU times reflect the speed of a 2.8Ghz Intel® Pentium® 4 processor with 768MB of RAM running *Mathematica* 5.0.

## ■ 2. Some Background: Solutions to Second-order Parabolic Partial Differential Equations as Path Integrals

This section outlines some well-known principles and relations on which the method developed in this paper is based. A complete, detailed, and rigorous description of these principles can be found in the ground-breaking work of Stroock and Varadhan [2] and also in [3, 4]. The computational aspects of the methods developed in these works have not been explored fully, and to a large extent the present work is a modest attempt to fill this gap.

The *Feynman integral* of a real function  $f[\cdot]$  defined on the space  $C_0[[t, T]]$ , which consists of continuous sample paths  $\omega : [[t, T]] \mapsto \mathbb{R}$  with the property  $\omega_t = 0$ , can be expressed as the following formal expression

$$\int_{\{\omega \in C_0[[t, T]]\}} f[\omega] e^{-\frac{1}{2} \int_t^T \left(\frac{d\omega_s}{ds}\right)^2 ds} \left( \prod_{s \in [[t, T]]} \frac{d\omega_s}{\sqrt{2\pi}} \right), \quad (2.1)$$

which is understood as the limit of the following sequence of multiple integrals

$$(2\pi)^{-\frac{k}{2}} \int_{\mathbb{R}} \dots \int_{\mathbb{R}} f_k[\omega_0, \omega_d, \dots, \omega_{kd}] e^{-\sum_{i=1}^k \frac{(\omega_{id} - \omega_{(i-1)d})^2}{2d}} \times d(\omega_d - \omega_0) \dots d(\omega_{kd} - \omega_{(k-1)d}), \quad (2.2)$$

where  $k \geq 1$ ,  $d := \frac{T-t}{k}$  and  $f_k[\omega_0, \omega_d, \dots, \omega_{kd}]$  is the result of the evaluation of  $f[\cdot]$  at the piecewise linear sample path from the space  $C_0[[t, T]]$ , whose values at the points  $t + id$ ,  $0 \leq i \leq k$ , are given by the vector  $(\omega_0, \omega_d, \dots, \omega_{kd})$ . It would not be an exaggeration to say that the convergence of the sequence in (2.2)—and therefore the fact that the integral in (2.1) is meaningful—is a miracle, without which very little of what we currently know about the universe would have taken place. Indeed, “miracle” is the only term that can explain the fact that, as was pointed out in [4], the infinities that arise under the exponent in (2.1) not only offset the infinities that arise under the product sign, but also offset them in such a way that the result is a perfectly well defined and highly nontrivial probability measure on the space  $C_0[[t, T]]$ , known as the *Wiener measure*. Not only is the integral in (2.1) the perfectly meaningful “expected value of  $f[\cdot]$ ,” but when  $f[\cdot]$  is given by  $f[\omega] := \Lambda[x + \omega_T]$ ,  $\omega \in C_0[[t, T]]$ , for some fixed  $x \in \mathbb{R}$  and some reasonably behaved function  $\Lambda: \mathbb{R} \mapsto \mathbb{R}$ , then, as a function of  $x$  and  $t$ , this expected value coincides with the solution of the standard heat equation

$$\partial_t u[t, x] + \mathcal{L}u[t, x] = 0, \quad \mathcal{L} := \frac{1}{2} \partial_{x,x}, \quad x \in \mathbb{R}, \quad t \leq T, \quad (2.3)$$

with boundary condition  $u[T, x] \equiv \Lambda[x]$ . In fact, this statement admits a generalization: if  $\mathcal{L}$  is replaced by a general elliptic differential operator of the form

$$\mathcal{L} = \frac{1}{2} \sigma^2[x] \partial_{x,x} + a[x] \partial_x, \quad (2.4)$$

assuming that the coefficients  $\sigma[\cdot]$  and  $a[\cdot]$  are reasonably behaved, the solution of the boundary value problem (2.3) can still be expressed as a path integral similar to the one in (2.1). The only difference is that in this case the path integral is understood as the limit of the sequence

$$\int_{\mathbb{R}} d(\omega_d - x) p_d[x, \omega_d] \times \int_{\mathbb{R}} d(\omega_{2d} - \omega_d) p_d[\omega_d, \omega_{2d}] \times \dots \times \int_{\mathbb{R}} d(\omega_{kd} - \omega_{(k-1)d}) p_d[\omega_{(k-1)d}, \omega_{kd}] \Lambda[x + \omega_{kd}], \quad k \geq 1, \quad (2.5)$$

where  $p_t[x, y]$  is the transition probability density of the Markov process  $X_t$ ,  $t \geq 0$ , determined by the stochastic equation

$$dX_t = \sigma[X_t] d\mathfrak{B}_t + a[X_t] dt, \quad (2.6)$$

in which  $\mathfrak{B}_t$ ,  $t \geq 0$ , is a given Brownian motion process. In other words,  $(t, x, y) \rightarrow p_t[x, y]$  is characterized by the fact that for any test function  $\varphi[\cdot]$  one has

$$E[\varphi[X_{t+b}] | X_t = x] = \int_{\mathbb{R}} \varphi[y] p_b[x, y] dy.$$

We will say that the process  $X_t$ ,  $t \geq 0$ , is a *sample path realization of the Markovian semigroup*  $e^{t\mathcal{L}}$ ,  $t \geq 0$ ,  $\mathcal{L}$  being the differential operator from (2.4), and will refer to the transition density  $(t, x, y) \rightarrow p_t[x, y]$  as *the integral kernel of the semigroup*  $e^{t\mathcal{L}}$ ,  $t \geq 0$ . This integral kernel is also known as *the fundamental solution of the equation*  $(\partial_t - \mathcal{L})u[t, x] = 0$ , that is, a solution which satisfies the initial condition  $u[0, x] = \delta_y[x]$ , where  $\delta_y[\cdot]$  stands for the usual Dirac delta function with mass concentrated at the point  $y \in \mathcal{D}$ . This means that, formally,  $p_t[x, y]$  can be expressed as  $p_t[x, y] = (e^{t\mathcal{L}} \delta_y)[x]$ .

A thorough exploration of this generalization of the Feynman integral can be found in the seminal work of Stroock and Varadhan [2]. The terminology is somewhat different from the one used here, that is, instead of “path integrals,” it comes down to a study of probability distributions on the pathspace associated with equations similar to the one in (2.6). Such a framework is much more rigorous and powerful. However, our goal is to develop a numerical procedure that computes the solution of the PDE in (2.3) as an expected value, and in this respect Feynman’s formalism proves to be quite useful.

### ■ 3. Path Integrals as Approximation Procedures

The multiple integral in (2.5) is actually an iterated integral, which can be transcribed as a recursive rule. More specifically, if for some fixed  $T \geq 0$ , the function  $(t, x) \rightarrow u[t, x]$  satisfies the equation

$$(\partial_t + \mathcal{L})u[t, x] = 0, \quad \mathcal{L} \equiv \frac{1}{2} \sigma^2[x] \partial_{x,x} + a[x] \partial_x,$$

for  $t \leq T$  and for  $x \in \mathbb{R}$ , assuming that the map  $x \rightarrow u[T, x]$  is given, one can compute the maps  $x \rightarrow u[T - id, x]$  consecutively for  $i = 1, 2, \dots$ , according to the following procedure

$$\begin{aligned} u[T - id, x] &= \int_{\mathbb{R}} p_d[x, y] \times u[T - (i - 1)d, y] dy \\ &= E[u[T - (i - 1)d, X_d] \mid X_0 = x], \quad i = 1, 2, \dots \end{aligned} \quad (3.1)$$

As we will soon see, the implementation of this procedure in *Mathematica* is completely straightforward. Since the above identity holds for every choice of  $d > 0$ , this raises the question: why does one need the recursive procedure at all? After all, one can always take  $d = T - t$  and then use the recursive rule in (3.1) only for  $i = 1$ , which comes down to what is known as the *Feynman–Kac formula*:

$$\begin{aligned} u[t, x] &= \int_{\mathbb{R}} p_{T-t}[x, y] \times u[T, y] dy \equiv E[u[T, X_{T-t}] \mid X_0 = x] \\ &\equiv E[\Lambda[X_{T-t}] \mid X_0 = x]. \end{aligned} \quad (3.2)$$

Unfortunately, the integral kernel  $(t, x, y) \rightarrow p_{T-t}[x, y] \equiv (e^{(T-t)\mathcal{L}} \delta_y)[x]$  can only be expressed in a form that allows one to actually compute this last integral—for example, by way of some numerical procedure—for some special choice of the coefficients  $\sigma[\cdot]$  and  $a[\cdot]$ . The idea, then, is to use the recursive rule

(3.1) with some “sufficiently small”  $d > 0$  and hope that for such a choice of  $d$ ,  $p_d[x, y]$  can be replaced with some approximate kernel, which, on the one hand, makes the integral (the expected value) on the right side of (3.1) computable and, on the other hand, does not allow the error in the calculation of  $u[T - id, x]$  caused by this approximation to grow larger than  $ido(d)$ . We will do what appears to be completely obvious and intuitive: replace  $\sigma[\cdot]$  and  $a[\cdot]$  with their first-order Taylor expansions about the point  $x \in \mathbb{R}$ . Intuitively, it should be clear that when the process  $X_t, t \geq 0$  is governed by the stochastic equation in (2.6) over a short period of time, its behavior cannot be very different from the behavior of a process that starts from the same position  $X_0 = x$  and is governed by the first-order Taylor approximation of the stochastic equation in (2.6) around the point  $x \in \mathbb{R}$ . The reason why one has to settle for a first-order approximation is that, in general, only when the coefficients  $\sigma[\cdot]$  and  $a[\cdot]$  in (2.6) are affine functions can one solve this equation explicitly, that is, write the solution as an explicit function of the driving Brownian motion. In general, the probability density of this explicit function is not something that is known to most computing systems and therefore its use in the computation of the integral on the right side of (3.1) is not straightforward. One way around this problem is to compute, once and for all, the transition probability density of a Markov process governed by a stochastic differential equation (SDE) with affine coefficients, and then add that function of seven variables (four parameters determined by the two affine coefficients  $\sigma[\cdot]$  and  $a[\cdot]$  plus the variables  $t, x$ , and  $y$ ) to the list of standard functions in the system. In this exposition we will not go that far, however. Instead, we will develop a second level of approximation, in which the solution of a general SDE with affine coefficients will be replaced by a computable function of the starting point  $x$ , the time  $t$ , and the position of the driving Brownian motion at time  $t$ . After this second level of approximation, the computation of the integral on the right side of (3.1) comes down to computing a Gaussian integral, which, as we will soon see, is not only feasible, but is also extremely easy to implement. Note that when the coefficients  $\sigma[\cdot]$  and  $a[\cdot]$  are linear functions, this second level of approximation of the solution of the stochastic equation in (2.6) actually coincides with the exact solution and is therefore redundant.

Before we can implement the plan that we just outlined, we must address an entirely mundane question: how is each function  $y \rightarrow u[T - id, y], i \geq 0$ , going to be represented in the computing system? The answer is that we will use polynomial interpolation; that is, in every iteration, the integral on the right side of (3.1) (with the approximate integral kernel) will be computed only for

$$x \in \mathcal{P} := \left\{ \Sigma + j \frac{b}{m}; -m \leq j \leq m \right\},$$

where  $\Sigma, b$ , and  $m$  are given parameters. Furthermore, in the actual computation of the integral, the map  $y \rightarrow u[T - (i - 1)d, y]$  will be replaced by the map constructed by way of polynomial interpolation (and extrapolation) from the values

$$u \left[ T - (i - 1)d, \Sigma + j \frac{b}{m} \right], \quad -m \leq j \leq m.$$

In other words, our strategy is to store each of the functions  $u[t, \cdot]$ , for  $t = T - id$ ,  $i = 1, 2, \dots$ , as a list of  $2m + 1$  real values and interpret  $u[t, \cdot]$  as the result of polynomial interpolation from that list. This means that we will be constructing the solution  $u[t, x]$  only for  $x \in [\Sigma - b, \Sigma + b]$  and  $t = T - id$ ,  $i = 1, 2, \dots$ , for some appropriate choice of  $d > 0$ . In most practical situations, one needs to construct  $u[t, \cdot]$  only on some finite interval and only for a fixed  $t < T$ . To do this with our method, one must set  $d = \frac{T-t}{k}$  for some sufficiently large  $k \in \mathbb{Z}_+$ , and then iterate (3.1)  $k$  times after choosing  $\Sigma$  and  $b$  so that the interval of interest is covered by the interval  $[\Sigma - b, \Sigma + b]$ ; the number of interpolation nodes ( $2m + 1$ ) must be chosen so that the polynomial interpolation is reasonably accurate. This might appear as a daunting programming task, but, as we will soon see, in *Mathematica* it involves hardly any programming at all. The reason is that the integration routine can accept as an input the output from the interpolation routine—all in symbolic form. This illustrates perfectly the point made earlier: new developments in the common man's computing tools can change the type of mathematics that one can use in practice.

A good point can and should be made that approaching (from a purely computational point of view) a general parabolic PDE like the one in (2.3) in the way we did and with the type of mathematics that we have chosen, namely, integration on the path space, is more than just natural. Indeed, the connection between the distribution of the sample paths of the process  $X_t$ ,  $t \geq 0$ , governed by the stochastic equation in (2.6) and the parabolic PDE (2.3) runs very deep (see [2, 3] for a detailed study of this relationship). The point is that the distribution on the space of sample paths associated with the SDE (2.6) encodes all the information about the elliptic differential operator  $\mathcal{L}$  from (2.4) and, therefore, all the information about the function

$$u[t, x] = (e^{(T-t)\mathcal{L}}) \Lambda[x], \quad x \in \mathbb{R}, \quad t \leq T.$$

This is analogous to the claim that the integral curves of a vector field encode all the information about the vector field itself. Actually, this is much more than a mere analogy. To see this, notice that if  $\mathcal{L}$  were a first-order differential operator (i.e., a vector field) in the open interval  $\mathcal{D} \subset \mathbb{R}$ , if  $t \rightarrow X_t$  were an integral curve for  $\mathcal{L}$  starting from  $X_0 = x$ , and if  $(t, x) \rightarrow u[t, x]$  has the property  $(\partial_t + \mathcal{L})u[t, x] = 0$ , then one would have

$$u[T, X_{T-t}] - u[t, x] = \int_t^T \partial_s u[s, X_{s-t}] ds = \int_t^T 0 ds = 0. \quad (3.3)$$

In other words, the integral curves of the vector field  $\mathcal{L}$  can be characterized as “curves along which any solution of  $(\partial_t + \mathcal{L})u[t, x] = 0$  remains constant.” This is interesting as well as quite practical. Indeed, if  $(\partial_t + \mathcal{L})u[t, x] = 0$  must be solved for  $t \leq T$  and  $x \in \mathcal{D}$  with boundary condition  $u[T, x] = \Lambda[x]$ ,  $x \in \mathcal{D}$ , one can compute the solution  $u[t, x]$  by first solving the ODE  $\frac{d}{dt} X_t = \mathcal{L}[X_t]$  with initial data  $X_0 = x$ . Here we treat  $\mathcal{L}$  as a “true” vector field (i.e., as a map of the form  $\mathcal{L}: \mathcal{D} \mapsto \mathbb{R}$ ) and then by evaluating the function  $\Lambda[\cdot]$  (assuming that it is given) at the point  $X_{T-t}$ . Thus, the computation of  $u[t, x]$ , for a given  $x \in \mathcal{D}$  and  $t < T$ ,

essentially comes down to constructing the integral curve  $[0, T - t] \ni s \rightarrow X_s$  for the vector field  $\mathcal{L}$  with initial position  $X_0 = x$ .

Of course, the remark just made can be found in any primer on PDEs and demonstrates how natural our approach to parabolic PDEs actually is. Indeed, the Feynman–Kac formula in (3.2) is essentially an analog of the relation in (3.3) in the case where  $\mathcal{L}$  is a second-order differential operator of elliptic type. This amounts to the claim that the solution of the parabolic PDE  $(\partial_t + \mathcal{L})u[t, x] = 0$  remains *constant on average* along the sample path of the process  $X_t$ ,  $t \geq 0$ , which is exactly the sample path representation of the semigroup  $e^{t\mathcal{L}}$ ,  $t \geq 0$ . Clearly, for any given starting point  $X_0 = x$ , the probability distribution of the sample path of the process  $X_t$ ,  $t \geq 0$ , has the meaning of a “distributed integral curve” for the second-order differential operator  $\mathcal{L}$ . This interpretation of the Feynman–Kac formalism is due to Stroock [3] and has been known for quite some time. The only “novelty” here is our attempt to utilize its computing power. The recursive rule (3.1) is analogous to a numerical procedure for constructing an integral curve of a standard (first-order) vector field by solving numerically a first-order ordinary differential equation.

One should point out that the main concept on which all finite difference methods rest is quite natural, too. Indeed, the intrinsic meaning of a partial derivative is nothing but “a difference quotient of infinitesimally small increments.” However, such a concept is intrinsic in general and not necessarily intrinsic for the equation  $(\partial_t + \mathcal{L})u[t, x] = 0$ . This is very similar to encoding an image of a circle as a list of a few thousand black or white pixels. There are various methods that would allow one to compress such an image, but they all miss the main point: a circle is determined only by its radius and center point, and this is the only data that one actually needs to encode. It is not surprising that, in order to achieve a reasonable accuracy, essentially all reincarnations of the finite difference method require a very dense grid on the time scale as well as the state-space.

As was pointed out earlier, path integrals can be used just as easily in the case of free boundary value problems of the following type: given an interval  $\mathcal{D} \subset \mathbb{R}$  and a number  $T \in \mathbb{R}_+$ , find a function  $]-\infty, T[ \times \mathcal{D} \ni (t, x) \rightarrow u[t, x] \in \mathbb{R}$  and also a free time-boundary  $]-\infty, T[ \ni t \rightarrow x^*[t] \in \mathcal{D}$  so that

- 1)  $u[t, x] \geq \Lambda[x]$ , for  $x \in \mathcal{D}$ ;
- 2)  $u[t, x] = \Lambda[x]$ , for  $x \in \mathcal{D} \cap ]-\infty, x^*[t][$ ;
- 3)  $(\partial_t + \mathcal{L})u[t, x] = 0$ , for  $x \in \mathcal{D} \cap ]x^*[t], \infty[$ ;
- 4)  $\lim_{x \searrow x^*[t]} u[t, x] = \Lambda[x^*[t]]$  and  $(\partial_x u)[t, x^*[t]] = \Lambda'[x^*[t]]$ ;
- 5)  $u[T, x] = \Lambda[x]$  for  $x \in \mathcal{D}$ .

This is an optimal stopping problem: at every moment  $t \leq T$ , one observes the value of the system process  $X_t$ ,  $t \geq 0$ , which is governed by the stochastic equation in (2.6), and decides whether to collect the termination payoff  $\Lambda[X_t]$  or take no action and wait for a better opportunity. The function  $(t, x) \rightarrow u[t, x]$ , which satisfies the conditions in (3.4), gives the price at time  $t$  ( $t < T$ ) of the right to collect the payoff at some moment during the time period  $[t, T]$ , assuming that at time  $t$  the observed value of the system process is  $X_t = x$ . Also, at time  $t$  ( $t < T$ ), the set  $]-\infty, x^*[t][ \cap \mathcal{D}$  gives the range of values of the system process

for which it is optimal to collect the payoff immediately, while the set  $]x^*[t], \infty[ \cap \mathcal{D}$  gives the range of values of the system process for which it is optimal to wait for a better opportunity. The valuation and the optimal exercise of an American stock option with payoff  $\Lambda[X_t]$ ,  $X_t$  being the price of the underlying asset at time  $t$ , is a classical example of an optimal stopping problem. Of course, here we assume that the exercise region has the form  $] - \infty, x^*[t]]$ . This assumption is always satisfied if  $\Lambda[\cdot]$  is a decreasing function of the system process, provided that larger payoffs are preferable. As it turns out, the recursive procedure (3.1) can produce, in the limit, the solution  $] - \infty, T] \times \mathcal{D} \ni (t, x) \rightarrow u[t, x]$  of the above optimal stopping problem with the following trivial modification:

$$u[T - i d, x] = \text{Max} \left[ \Lambda[x], \int_{\mathbb{R}} p_d[x, y] \times u[T - (i - 1) d, y] d y \right], \tag{3.5}$$

$$i = 1, 2, \dots$$

This is simply a rephrasing of Bellman’s principle for optimality in discrete time, and the conditions in (3.4) are simply a special case of Bellman’s equation.

We conclude this section with the remark that the above considerations can easily be adjusted to the case where the elliptic differential operator  $\mathcal{L}$  contains a 0-order term, that is,

$$\mathcal{L} = \frac{1}{2} \sigma^2[x] \partial_{x,x} + a[x] \partial_x - r \tag{3.6}$$

where  $r \in \mathbb{R}$  is a given parameter. The reason why such a generalization is actually trivial is that when  $(t, x) \rightarrow u[t, x]$  satisfies the equation  $(\partial_t + \mathcal{L}) u[t, x] = 0$  with  $\mathcal{L} = \frac{1}{2} \sigma^2[x] \partial_{x,x} + a[x] \partial_x$ , then the function

$$(t, x) \rightarrow e^{-(T-t)r} u[t, x]$$

satisfies the same equation, but with  $\mathcal{L} = \frac{1}{2} \sigma^2[x] \partial_{x,x} + a[x] \partial_x - r$ . In any case, if  $\mathcal{L}$  is chosen as in (3.6) and not as in (2.4), the only change that one will have to make in the procedure outlined earlier in this section would be to replace the recursive procedure in (3.1) and (3.5), respectively, with

$$u[T - i d, x] = \int_{\mathbb{R}} e^{-r d} p_d[x, y] \times u[T - (i - 1) d, y] d y,$$

$$i = 1, 2, \dots,$$

and

$$u[T - i d, x] = \text{Max} \left[ \Lambda[x], \int_{\mathbb{R}} e^{-r d} p_d[x, y] \times u[T - (i - 1) d, y] d y \right],$$

$$i = 1, 2, \dots$$

## ■ 4. The Black–Scholes Formula versus the Feynman–Kac Representation of the Solution of the Black–Scholes PDE

In the option pricing model developed by F. Black and M. Scholes [1] the asset price  $S$  is assumed to be governed by the stochastic equation

$$dS_t = \varsigma S_t d\mathbb{B}_t + \mu S_t dt, \quad t \geq 0, \quad (4.1)$$

in which  $\varsigma > 0$  and  $\mu > 0$  are given parameters. At time  $t > 0$  the price of an European call option with payoff  $\Lambda[S] = \text{Max}[S - K, 0]$ , which expires at time  $T > t$ , can be identified with the expression  $u[t, S_t]$ , where  $(t, x) \rightarrow u[t, x]$  is the solution of the PDE:

$$\begin{aligned} \partial_t u[t, x] + \frac{1}{2} \sigma^2 x^2 \partial_x^2 u[t, x] + rx \partial_x u[t, x] - ru[t, x] = 0, \\ t \leq T, \quad x > 0, \end{aligned} \quad (4.2)$$

with boundary condition  $\lim_{t \nearrow T} u[t, x] = \Lambda[x]$ , where  $r > 0$  is a parameter representing the (constant) short-term interest rate and  $\Lambda[\cdot]$  gives the termination payoff as a function of the price of the underlying asset. The fact that the option price must satisfy the PDE (4.2) may be seen as a consequence of Bellman's principle for optimality, but one has to keep in mind that the use of this principle in finance is somewhat subtle. Notice, for example, that the Feynman–Kac formula (3.2) links the PDE (4.2) not with the stochastic equation in (4.1), but, rather, with the stochastic equation

$$dX_t = \varsigma X_t d\mathbb{B}_t + r X_t dt, \quad t \geq 0. \quad (4.3)$$

Therefore, the solution of the Black-Scholes equation in (4.2) can be expressed as an expected value of the form

$$u[t, x] = E[e^{-r(T-t)} \text{Max}[X_{T-t} - K, 0] \mid X_0 = x], \quad t < T, \quad x \in \mathbb{R}_+,$$

and *not* as

$$u[t, x] = E[e^{-r(T-t)} \text{Max}[S_{T-t} - K, 0] \mid S_0 = x], \quad t < T, \quad x \in \mathbb{R}_+.$$

In other words, in the context of option pricing, one must use Bellman's principle as if the predictable part in the system process has the form  $r X_t dt$ , regardless of the form that the predictable part in the system process actually has. This phenomenon is well known and its justification, of which there are several, can be found in virtually any primer on financial mathematics. Roughly speaking, this is caused by the fact that one must take into account the reward on average which investors collect from the underlying asset (see [5]).

Since the stochastic equation in (4.3) admits an explicit solution given by

$$X_t = X_0 e^{\varsigma \mathbb{B}_t + (r - \frac{1}{2} \varsigma^2)t}, \quad t \geq 0,$$

and since this solution is simple enough, it is possible to implement the recursive procedure from (3.1) with a single iteration. This means that, in this special case, the Feynman–Kac formula in (3.2) is directly computable without the need to approximate the integral kernel of the associated Markovian semigroup. This

also eliminates the need for interpolation, if all that one wants to do is compute the solution  $u[t, x]$  for some fixed  $t$  and some fixed  $x$ . Of course, this assumes that the payoff can be expressed in a way that *Mathematica* can understand. In our first example, we will compute  $u[0, 50]$ , assuming that  $\varsigma = 0.2$ ,  $r = 0.1$  and that the boundary condition is  $u[3, x] = \Lambda[x]$ , where  $\Lambda[x] = \text{Max}[x - 40, 0]$ .

```

In[1]:= T = 3.0; \varsigma = 0.2; r = 0.1; \Lambda[x_] := Max[x - 40, 0];
In[2]:= OptionPrice[S_] :=
      \frac{e^{-T \times r}}{\sqrt{2 \pi}} NIntegrate[\Lambda[S \times e^{(r - \frac{\varsigma^2}{2}) \times T + \varsigma \sqrt{T} y}] \times e^{-\frac{y^2}{2}}, \{y, -\infty, \infty\},
      MaxRecursion -> 15, WorkingPrecision -> 20, PrecisionGoal -> 6]

```

Here is the result.

```

In[3]:= OptionPrice[50] // Timing
Out[3]= {0.13 Second, 20.7449}

```

What we have just computed was the price of an European call option with a strike price of \$40 and  $T = 3$  years to maturity, assuming that the current price of the underlying asset is \$50, that the volatility in the stock price is 20 percent/year and that the fixed interest rate is 10 percent/year. Of course, the price of such an option can be computed directly by using the Black–Scholes formula. First, we will express the Black–Scholes formula in symbolic form

$$\begin{aligned}
 \text{In[4]:= } \mathcal{N}[x_] &:= \frac{1}{2} \text{Erf}\left[\frac{x}{\sqrt{2}}\right] + \frac{1}{2}; \\
 \text{BS}[r_-, \varsigma_-, x_-, K_-, T_] &:= x \times \mathcal{N}\left[\left(\text{Log}\left[\frac{x}{K}\right] + \left(r + \frac{1}{2} \varsigma^2\right) \times T\right) / (\varsigma \sqrt{T})\right] - \\
 &K \times e^{-r \times T} \mathcal{N}\left[\left(\text{Log}\left[\frac{x}{K}\right] + \left(r - \frac{1}{2} \varsigma^2\right) \times T\right) / (\varsigma \sqrt{T})\right]
 \end{aligned}$$

and then compute the price of the same option.

```

In[5]:= BS[r, \varsigma, 50, 40, T] // Timing
Out[5]= {0. Second, 20.7449}

```

One can see from this example that, from a practical point of view, the Feynman–Kac representation of the solution of the Black–Scholes PDE in *Mathematica* is just as good as the “explicit” Black–Scholes formula. (Ironically, the Black–Scholes formula takes somewhat longer to type, not to mention the time it might take to retrieve it from a book.) However, the Feynman–Kac representation is much more universal. Indeed, with no extra work—and no additional complications whatsoever—it allows one to compute the value of an option with a very general payoff. Consider for example an option that pays at maturity the smallest of the payoffs from a call with a strike price of \$40 and a put with a strike price of \$80.

```

In[6]:= \Lambda[x_] := Min[Max[x - 40, 0], Max[80 - x, 0]]

```

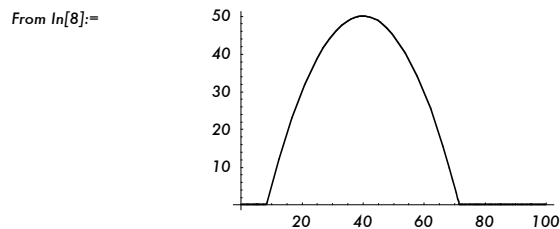
With all other parameters kept the same as before (3 years left to maturity, 20 percent volatility and 10 percent interest rate), assuming that the stock price is \$50, the price of this option is

```
In[7]:= OptionPrice[50] // Timing
```

```
Out[7]= {0.031 Second, 5.22782}
```

Obviously, the computing time is negligible, but what is much more important is that the *only* thing that we had to do in this case was to simply change the definition of the payoff. Here is another example.

```
In[8]:=  $\Lambda[x_] := \text{Max}[50 - 0.05(x - 40)^2, 0]$ ; Plot[\Lambda[x], {x, 0, 100}]
```



```
In[9]:= OptionPrice[50] // Timing
```

```
Out[9]= {0.015 Second, 16.5291}
```

We will conclude this section with an example of what is known as a cash-or-nothing option.

```
In[10]:=  $\Lambda[x_] := 20 \times \text{Max}[\text{Sign}[x - 40], 0]$ 
```

```
In[11]:= Off[NIntegrate::"slwcon"]; OptionPrice[50] // Timing
```

```
Out[11]= {0.031 Second, 13.4738}
```

The overwhelming majority of the options that are currently traded are some combination of calls, puts, and cash-or-nothing options. The origins of this tradition are not exactly certain, but it might have something to do with the fact that these are the options for which popular and easy-to-use computing tools have been widely available. It should be clear from the examples presented in this section that, in the realm of the Black–Scholes model, essentially any reasonably defined option is, in fact, trivial to price, and that having a Black–Scholes formula is no longer necessary. Of course, the well-known binomial model does allow one to compute the value of an option with a very general payoff; however, this popular approach does require some work, and its accuracy and overall efficiency is far from satisfactory. Furthermore, this method is tied exclusively to the assumption that stock prices follow the law of a geometric Brownian motion; the technology that we are developing is not.

One must realize that, in keeping with the method outlined in Section 3, solving the PDE in (4.2) is an entirely trivial matter. The point is that the integral kernel of the associated Markovian semigroup happens to be directly computable, in the sense that it can be written in symbolic form that can be understood by *Mathemat-*

ica. However, our main goal is to demonstrate that this same method is just as easy to implement in much more general situations. In terms of applications to computational finance, that would mean that considerably more sophisticated models for asset values—for example, models with stochastic volatility and stochastic return of a rather general form—would be just as easy to use as the basic Black–Scholes model. The main step in this program is to develop an adequate approximation of the sample path representation of the semigroup  $e^{t\mathcal{L}}$ ,  $t \geq 0$ , for a general elliptic differential operator  $\mathcal{L}$  (see the next section).

### ■ 5. Approximation of the Fundamental Solution by Way of a Discrete Dynamic Programming Procedure

In general, there are many ways in which the solution of a given SDE can be approximated by a discrete time series (see [6] for a thorough exploration and a comprehensive list of references on this topic). Unfortunately, none of the available approximation schemes was developed for a purpose similar to ours, so we will develop one which actually is. Another extremely interesting approximation method, whose computational aspects are yet to be explored, can be found in [7].

To begin with, notice that the stochastic equation in (2.6) can be written in integral form as

$$X_t = x + \int_0^t \sigma[X_s] d\beta_s + \int_0^t a[X_s] ds, \quad X_0 = x \in \mathbb{R}, \quad t \geq 0. \quad (5.1)$$

One possible approximation of this equation can be obtained by replacing the coefficients  $\sigma[\cdot]$  and  $a[\cdot]$  by constants, which gives the following SDE

$$Y_t = x + \int_0^t \sigma[x] d\beta_s + \int_0^t a[x] ds, \quad Y_0 = x \in \mathbb{R}^n.$$

This approximation, which we will call the 0<sup>th</sup>-order approximation, is so crude that the SDE it leads to admits a trivial solution:

$$Y_t = x + \sigma[x] \beta_t + a[x] t, \quad t \geq 0.$$

Intuitively, it should be clear that, when  $t$  is sufficiently small, the distribution of the random variable (r.v.)  $Y_t$  is not very far from that of the r.v.  $X_t$ . If we were to implement the plan outlined in Section 3 with this level of approximation of the Markovian semigroup associated with the equation  $(\partial_t + \mathcal{L})u[t, x] = 0$ , the recursive procedure in (3.1) would come down to

$$\begin{aligned} & u\left[T - i \frac{T-t}{k}, x\right] \\ &= (2\pi)^{-\frac{1}{2}} \int_{\mathbb{R}} u\left[T - (i-1) \frac{T-t}{k}, x + \sigma[x] \frac{T-t}{k} \gamma + a[x] \frac{T-t}{k}\right] e^{-\frac{\gamma^2}{2}} d\gamma, \\ & \hspace{15em} i = 1, \dots, k. \end{aligned}$$

Assuming that  $u[T - (i-1) \frac{T-t}{k}, \cdot]$  is the result of polynomial interpolation, the above integral is certainly computable for every choice of  $x$ : in fact, it only has to

be computed for  $x \in \mathcal{P}$ , where  $\mathcal{P}$  stands for the set of interpolation nodes. One can show that as  $k \rightarrow \infty$  and as the number of interpolation nodes increases to  $\infty$ ,  $u[T-t, \cdot]$  converges to the actual solution uniformly in the interpolation region. However, this convergence is known to be rather slow. The 0<sup>th</sup>-order approximation of SDEs is as old as the theory of SDEs itself and can be traced back to the original works of Itô—in fact, this approximation is crucial for his construction of the stochastic integral. One may say that the 0<sup>th</sup>-order approximation is “optimal” if all that one needs to do is show the existence of solutions to SDEs and derive various properties of the associated probability distributions, in which case the speed of the convergence is not an issue. In our case, however, it is and one can do considerably better by using the 1<sup>st</sup>-order Taylor approximation of the coefficients  $\sigma[\cdot]$  and  $a[\cdot]$ . Our plan, then, is to replace the stochastic equation in (5.1) with the following SDE

$$Z_t = x + \int_0^t \left( \sigma[x] + \sigma'[x](Z_s^x - x) \right) d\mathbb{B}_s + \int_0^t \left( a[x] + a'[x](Z_s^x - x) \right) ds, \quad t \geq 0,$$

which admits this explicit solution

$$\begin{aligned} Z_t = & e^{\sigma'[x]\mathbb{B}_t + (a'[x] - \frac{1}{2}\sigma'[x]^2)t} \\ & \times \left( (a[x] - a'[x]x) \int_0^t e^{-\sigma'[x]\mathbb{B}_s - (a'[x] - \frac{1}{2}\sigma'[x]^2)s} ds \right. \\ & \quad \left. + (\sigma[x] - \sigma'[x]x) \int_0^t e^{-\sigma'[x]\mathbb{B}_s - (a'[x] - \frac{1}{2}\sigma'[x]^2)s} d\mathbb{B}_s + x \right) \\ & - \sigma'[x] \times (\sigma[x] - \sigma'[x]x) \times e^{\sigma'[x]\mathbb{B}_t + (a'[x] - \frac{1}{2}\sigma'[x]^2)t} \\ & \times \int_0^t e^{-\sigma'[x]\mathbb{B}_s - (a'[x] - \frac{1}{2}\sigma'[x]^2)s} ds. \end{aligned} \quad (5.2)$$

In spite of the “explicit” form of this solution, the density of the r.v.  $Z_t$  is still difficult to compute. Therefore, we need to develop yet another level of approximation. In order to approximate the random variable in the right side of (5.2) with something computable, notice that if the integrand in the first and the third integral is replaced by the constant 1 this will produce an error of order  $o(t)$  (a *small*  $o$  of  $t$ ). The approximation of the second integral is somewhat more involved. A straightforward application of the Itô formula yields

$$\begin{aligned} & \int_0^t e^{-\sigma'[x]\mathbb{B}_s - (a'[x] - \frac{1}{2}\sigma'[x]^2)s} d\mathbb{B}_s \\ & = \frac{1}{\sigma'[x]} \left( 1 - e^{-\sigma'[x]\mathbb{B}_t - (a'[x] - \frac{1}{2}\sigma'[x]^2)t} \right) \\ & \quad + \left( \sigma'[x] - \frac{a'[x]}{\sigma'[x]} \right) \int_0^t e^{-\sigma'[x]\mathbb{B}_s - (a'[x] - \frac{1}{2}\sigma'[x]^2)s} ds, \end{aligned}$$

which shows that

$$\begin{aligned} & \int_0^t e^{-\sigma'[x]\mathbb{B}_s - (a'[x] - \frac{1}{2}\sigma'[x]^2)s} d\mathbb{B}_s \\ & = \frac{1}{\sigma'[x]} \left( 1 - e^{-\sigma'[x]\mathbb{B}_t - (a'[x] - \frac{1}{2}\sigma'[x]^2)t} \right) + \sigma'[x]t - \frac{a'[x]}{\sigma'[x]}t + o(t). \end{aligned}$$

Thus, (5.2) can be rewritten as

$$Z_t = e^{\sigma'[x] \mathcal{B}_t + (a'[x] - \frac{1}{2} \sigma'^2[x])t} \times \left( (a[x] - a'[x]x)t + (\sigma[x] - \sigma'[x]x) \right. \\ \left. \times \left( \frac{1}{\sigma'[x]} \left( 1 - e^{-\sigma'[x] \mathcal{B}_t - (a'[x] - \frac{1}{2} \sigma'^2[x])t} \right) - \frac{a'[x]}{\sigma'[x]} t \right) + x \right) + o(t).$$

Of course, this identity holds only if  $\sigma'[x] \neq 0$ , but this is the only truly interesting case, since  $\sigma'[x] = 0$  implies that

$$Z_t = e^{a'[x]t} \times \left( (a[x] - a'[x]x)t + \sigma[x] \int_0^t e^{-a'[x]s} d\mathcal{B}_s + x \right) + o(t).$$

The stochastic integral in this expression is a Gaussian r.v. of variance  $\frac{1}{2a'[x]} (1 - e^{-2a'[x]t}) = t - a'[x]t^2 + O(t^3)$ . Thus, we will approximate expressions of the form  $E[f[Z_t]]$  with

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f[\psi[t, x, \gamma]] e^{-\frac{\gamma^2}{2}} d\gamma \tag{5.3}$$

where

$$\ln[12] := \varepsilon = 0.0001; \psi[t, x, \gamma] := \text{If}[\text{Abs}[\sigma'[x]] > \varepsilon, \\ e^{\sigma'[x] \times \gamma \times \sqrt{t} + (a'[x] - \frac{1}{2} \sigma'^2[x]) \times t} \times \left( (a[x] - a'[x] \times x) \times t + (\sigma[x] - \sigma'[x] \times x) \times \right. \\ \left. \left( \frac{1}{\sigma'[x]} \left( 1 - e^{-\sigma'[x] \times \gamma \times \sqrt{t} - (a'[x] - \frac{1}{2} \sigma'^2[x]) \times t} \right) - \frac{a'[x]}{\sigma'[x]} \times t \right) + x \right), \\ e^{a'[x] \times t} \times \left( (a[x] - a'[x] \times x) \times t + \sigma[x] \times \gamma \times \sqrt{t - a'[x] \times t^2} + x \right)]$$

In particular, the recursive procedure in (3.1) will be replaced with

$$u\left[T - i \frac{T-t}{k}, x\right] \\ = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u\left[T - (i-1) \frac{T-t}{k}, \psi\left[\frac{T-t}{k}, x, \gamma\right]\right] e^{-\frac{\gamma^2}{2}} d\gamma, \\ i = 1, \dots, k,$$

in the case of a terminal value problem, and with

$$u\left[T - i \frac{T-t}{k}, x\right] \\ = \text{Max}\left[\Lambda[x], \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u\left[T - (i-1) \frac{T-t}{k}, \psi\left[\frac{T-t}{k}, x, \gamma\right]\right] e^{-\frac{\gamma^2}{2}} d\gamma\right], \\ i = 1, \dots, k,$$

in the case of a free boundary value problem. In Section 6 we will show that the computation in *Mathematica* of the last two integrals for a fixed  $x \in \mathcal{P}$  is com-

pletely straightforward when the object  $u[T - (i - 1) \frac{T-t}{k}, \cdot]$  is the result of polynomial interpolation.

It is not hard to check that in the special case where  $\sigma[x] \equiv \varsigma \times x$  and  $a[x] \equiv r \times x$  one has

$$\psi[\Delta, x, y] = x e^{\varsigma y \sqrt{\Delta} + (r - \frac{1}{2} \varsigma^2) \Delta},$$

which means that, with this choice for  $\sigma[\cdot]$  and  $a[\cdot]$ , the solution to the stochastic equation in (5.1) is simply  $X_t = \psi[t, x, \mathfrak{B}_t]$ .

## ■ 6. Dynamic Programming with *Mathematica*: The Case of American and European Stock Options

First, we will consider several examples of terminal-value problems of the form:

$$(\partial_t + \mathcal{L}) u[t, x] = 0, \quad x \in \mathbb{R}, \quad t \leq T, \quad u[T, x] = \Lambda[x] \quad (6.1)$$

where  $\mathcal{L}$  stands for the differential operator  $\frac{1}{2} \sigma^2[x] \partial_{x,x} + a[x] \partial_x - r$ . The method that we developed in Sections 3 and 5 can be implemented in *Mathematica* with the following three definitions (in addition to the definition of  $\psi[t, x, \gamma]$  from Section 5), in which the interpolation region is the interval  $[\Sigma - b, \Sigma + b]$ , the number of (equally spaced) interpolation nodes is  $2m + 1$ , and the time step is  $\frac{T-t}{k}$ . The object  $U[i, \cdot, k, \cdot, \cdot, \cdot]$  represents the list of all  $2m + 1$  values of  $u[T - i \frac{T-t}{k}, \cdot]$  at the entire set of interpolation nodes  $\mathcal{P}$  and, for a given object  $f$ , which *Mathematica* treats as a function,  $A[f, \cdot, \cdot, \cdot, \cdot]$  returns the list of values of  $E[f[\psi[d, x, \gamma]]]$  for all  $x \in \mathcal{P}$ .

$$\text{In[13]:= } U[0, T_, k_, \Sigma_, h_, m_] := \text{Table}[\Lambda[\Sigma + j \times \frac{b}{m}], \{j, -m, m\}]$$

$$\text{In[14]:= } U[i, T_, k_, \Sigma_, h_, m_] :=$$

$$\left( f[x_] = \text{ListInterpolation}[U[i - 1, T, k, \Sigma, b, m], \{\{\Sigma - b, \Sigma + b\}\}][x];$$

$$\text{Table}[\frac{e^{-r \times \frac{T}{k}}}{\sqrt{2\pi}} \times \text{NIntegrate}[f[\psi[\frac{T}{k}, \Sigma + j \times \frac{b}{m}, y]] e^{-\frac{y^2}{2}},$$

$$\{y, -7, 7\}, \text{MaxRecursion} \rightarrow 20], \{j, -m, m\}] \right)$$

Note that we have replaced  $\int_{-\infty}^{\infty}$  with  $\int_{-7}^7$ . This makes the integration procedure more efficient, while the loss in precision is well within the accuracy of the method. One must be aware, however, that the interval  $[-7, 7]$  may be too narrow for some problems.

Since our goal here is not to improve *Mathematica*'s performance, we will ignore the following error messages.

```
In[15]:= Off[InterpolatingFunction::"dmval"]; Off[InterpolatingFunction::"dmvali"];
Off[NIntegrate::"ploss"]; Off[NIntegrate::"ncvb"];
Off[NIntegrate::"slwcon"]; Off[InverseFunction::"ifun"];
Off[Power::"infy"]; Off[$MaxExtraPrecision::"meprec"]; Off[∞::"indet"]
```

In our first example, we take

```
In[16]:= Λ[x_] := Max[x - 80, 0]; r = 0.1; σ[x_] := 0.2 × x; a[x_] := r × x;
```

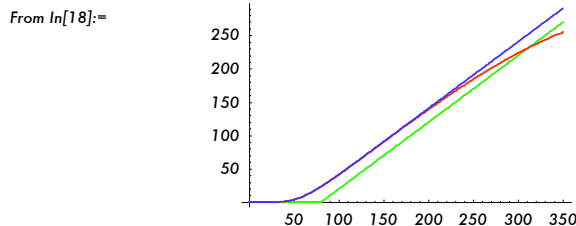
The solution  $u[0, x]$  gives the value of an European call option with a strike price of \$80 that expires in  $T$  years, assuming that the volatility of the stock is 20 percent/year and the interest rate is 10 percent/year. We will take  $T = 3$  years to maturity,  $k = 5$  time periods (5 iterations), and 61 interpolation nodes ( $m = 30$ ) in the interpolation interval  $[[80 - 75, 80 + 75]]$ .

```
In[17]:= μ = U[5, 3, 5, 80, 75, 30] // Timing; μ[1]
```

```
Out[17]= 2.172 Second
```

We can compare the result (the red curve) with the “exact” solution (the blue curve) produced by the Black–Scholes formula; the green curve represents the termination payoff.

```
In[18]:= Plot[Λ[x], ListInterpolation[μ[2], {{80 - 75, 80 + 75}}][x],
BS[0.1, 0.2, x, 80, 3], {x, 0, 350}, PlotPoints -> 200,
PlotRange -> All, PlotStyle -> {Hue[0.3], Hue[0.], Hue[.7]}]
```



One can see that the numerical solution (the red curve) is reasonably close to the exact solution in most of the interpolation region. To develop some sense for the accuracy of the method, we will compare the value of the option when the stock price is \$50, produced by the dynamic programming procedure

```
In[19]:= ListInterpolation[μ[2], {{80 - 75, 80 + 75}}][50]
```

```
Out[19]= 3.7509
```

with the following value produced by the Black–Scholes formula.

```
In[20]:= BS[0.1, 0.2, 50, 80, 3]
```

```
Out[20]= 3.75529
```

Of course, one can also compute this value by using the Feynman–Kac formula, just as we did in Section 4.

```
In[21]:= r = 0.1; σ = 0.2; OptionPrice[50]
```

```
Out[21]= 3.75529
```

As was pointed out earlier, when the coefficients are linear,  $\psi[]$  gives the exact solution and one can get away with a single iteration, that is, with a time step equal to the entire time interval. Essentially this comes down to using the Feynman–Kac formula directly.

```
In[22]:=  $\mu = U[1, 3, 1, 80, 75, 30]; \text{ListInterpolation}[\mu, \{\{80 - 75, 80 + 75\}\}][50]$ 
```

```
Out[22]= 3.75083
```

The only reason the last two results are different is that  $U[]$  applies the Feynman–Kac formula to the interpolated payoff, while  $\text{OptionPrice}[]$  applies the Feynman–Kac formula to the actual payoff. Of course, there is no reason to use the dynamic programming procedure when the coefficients are linear and one deals with a fixed boundary value problem; here we did so only for the purpose of illustration.

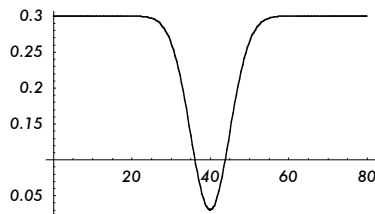
Now we will consider an example in which the coefficients  $\sigma[]$  and  $a[]$  are not linear, and therefore the Feynman–Kac formula cannot be used directly, that is, with a single iteration.

```
In[23]:=  $\sigma[x_] := 0.3 (1 - 0.9 e^{-0.02(40-x)^2}) \times x; r = 0.1;$   
 $a[x_] := r \times x; \Lambda[x_] := \text{Max}[x - 40, 0]$ 
```

With this choice of the coefficients  $\sigma[]$  and  $a[]$  and the terminal data  $\Lambda[]$ , the solution of the PDE in (6.1) will give the price of a call option with a strike price of \$40, but with stochastic volatility for the underlying asset. For the purpose of illustration, we choose a volatility coefficient that depends on the price of the stock: it decreases to 0.1 when the price of the stock approaches the strike price of the option and stays constant at 0.2 when the stock price is away from the strike price.

```
In[24]:=  $\text{Plot}[\frac{\sigma[x]}{x}, \{x, 0.05, 80\}, \text{PlotRange} \rightarrow \text{All}, \text{PlotPoints} \rightarrow 200]$ 
```

```
From In[24]:=
```

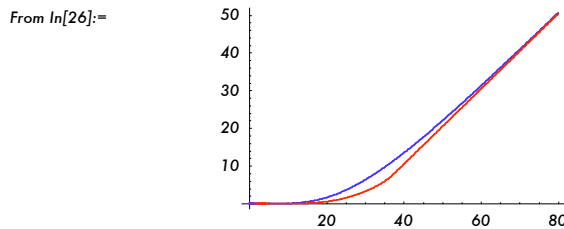


```
In[25]:=  $\text{mk} = U[20, 3, 20, 50, 48, 50] // \text{Timing}; \text{mk}[1]$ 
```

```
Out[25]= 14.828 Second
```

Now we can compare the price of this option with the one produced by the Black–Scholes formula (the blue line), which ignores the fact that the volatility decreases when the stock price approaches the strike price. Just as one would expect, when the stock price is away from the region where the volatility decreases, the price of the option looks the same as the price produced under the assumption of constant volatility.

```
In[26]:= Plot[BS[0.1, 0.3, x, 40, 3], ListInterpolation[mk[2], {{50 - 48, 50 + 48}}][x],
{x, 0, 80}, PlotPoints -> 400, PlotStyle -> {Hue[0.7], Hue[0.]}]
```



When the stock price is \$50, the price of this option is

```
In[27]:= ListInterpolation[mk[2], {{50 - 48, 50 + 48}}][50]
```

Out[27]= 20.4144

Notice that we have chosen the interpolation interval  $[[50 - 48, 50 + 48]]$  so that it is centered around the stock price of interest, namely \$50; remember that the accuracy of the method is the highest at the center of the interpolation interval. Notice also that, while it took much longer to compute, the calculation of the option price under stochastic volatility did not require any additional work.

Our next example, which is unrelated to finance, will compute the solution  $x \rightarrow u[0, x]$  of the equation

$$\partial_t u[t, x] + e^{-x^2} \partial_{x,x} u[t, x] + \text{Sin}[x] \partial_x u[t, x] = 0$$

with boundary condition  $u[3, x] = \text{Sign}[x]$ .

```
In[28]:= σ[x_] := √2 e-x2/2; a[x_] := Sin[x]; Λ[x_] := Sign[x]; r = 0;
```

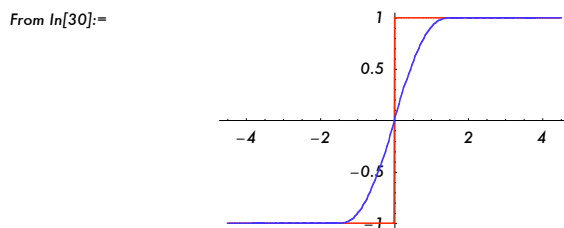
The interpolation interval is  $[[0 - 5, 0 + 5]]$ , the number of interpolation nodes is  $2 \times 25 + 1 = 51$ , and the time step is  $\frac{3}{30} = 0.1$ .

```
In[29]:= g0 = U[30, 3., 30, 0, 5, 25] // Timing; g0[1]
```

Out[29]= 10.656 Second

Now we can compare the solution that we just found, that is, the function  $u[0, \cdot]$  (pictured as the blue line below) with the terminal data  $u[3, \cdot]$  (pictured as the red line).

```
In[30]:= Plot[{Λ[x], ListInterpolation[g0[2], {{-5, 5}}][x]},
{x, -4.5, 4.5}, PlotStyle -> {Hue[0], Hue[.7]}, PlotPoints -> 200]
```



Sometimes one must use the dynamic programming procedure in a nontrivial way and work with some small time step, even if the coefficients  $\sigma[]$  and  $a[]$  happen to be linear; such a case is an optimal stopping problem, which leads to boundary conditions prescribed along a free boundary. Now we will consider several examples that involve stock options in which early exercise is possible (the so-called American options). We only need to change the definition of the function  $U[]$ ; everything else in the procedure will remain the same.

```
In[31]:= U[i_, T_, k_, Σ_, h_, m_] :=
  ( f[x_] = ListInterpolation[U[i - 1, T, k, Σ, h, m], {{Σ - b, Σ + b}}][x]; Table[
    Max[Λ[Σ + j ×  $\frac{b}{m}$ ],  $\frac{e^{-r \times \frac{T}{k}}}{\sqrt{2 \pi}}$  × NIntegrate[f[ψ[ $\frac{T}{k}$ , Σ + j ×  $\frac{b}{m}$ , y]] e $^{-\frac{y^2}{2}}$ ,
      {y, -7, 7}, MaxRecursion → 20]], {j, -m, m} ] )
```

Under the Black–Scholes assumption for the price process, American call options are the same as European call options because early exercise of such options is never optimal. Thus, in our first example we consider an American put option with a strike price of \$40.

```
In[32]:= Λ[x_] := Max[40 - x, 0]
```

First, we will compute the price of the option under the assumption that—just as in the Black–Scholes model—the stock price follows a geometric Brownian motion process (of course, the Black–Scholes formula cannot be used in this case as is, for it is only valid for European options).

```
In[33]:= r = 0.1; σ[x_] := 0.2 × x; a[x_] := r × x;
```

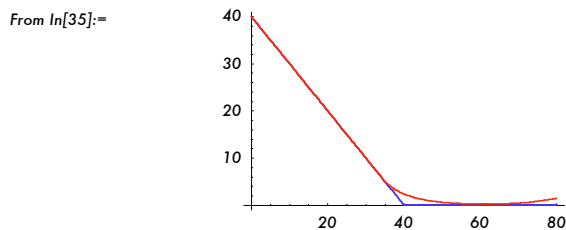
We will interpolate the solution in the interval  $[[40 - 38, 40 + 38] \equiv [2, 78]]$  with 41 interpolation nodes and use 20 iterations, which amounts to a time step of 0.15.

```
In[34]:= m = U[20, 3, 20, 40, 38, 20] // Timing; m[[1]]
```

```
Out[34]= 6.438 Second
```

Now we can plot the solution (the red curve below) together with the termination payoff function (the blue curve).

```
In[35]:= Plot[{Λ[x], ListInterpolation[m[[2]], {{40 - 38, 40 + 38}}][x]},
  {x, 0, 80}, PlotPoints → 200, PlotStyle → {Hue[0.7], Hue[0.]}]
```

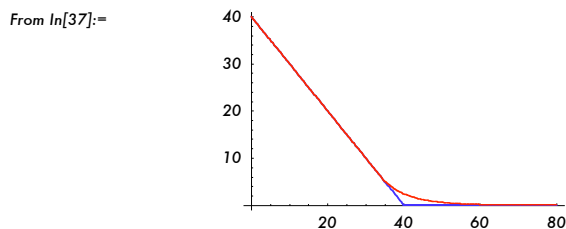


To those familiar with the basics of American options, it should be clear from this graph that the approximation that we just produced is not very accurate in the last 1/4 of the interpolation interval (it is well known that the actual stock price decreases to 0 monotonically). This is caused by the fact that, when the process  $X_t$ ,  $t \geq 0$ , starts from a point that is close to the end of the interpolation interval, a sizeable part of the distribution of  $X_t$  might be spread in the region where the procedure is forced to use extrapolation. As a result, depending on how far inside the extrapolation region the essential part of the distribution of  $X_t$  goes, the error from the extrapolation can be significant. In this particular example, one possible remedy would be to set  $u[T - (i - 1)\frac{T-t}{k}, x]$  to 0 for all sufficiently large values of  $x$ , regardless of the value produced by the interpolation (or the extrapolation) procedure. This is a common problem in essentially all numerical methods for PDEs, and the common way around it is to work with a grid in the state-space which covers a considerably larger region than the one in which an accurate representation of the solution is being sought. It is to be noted, however, that, as  $k \rightarrow \infty$ , that is, as the time step converges to 0, the solution produced after  $k$  iterations will converge to the actual solution in the *entire* interpolation interval. This is because when the process  $X_t$ ,  $t \geq 0$ , starts from one of the end-points of that interval, for a very small value of  $t$ , the distribution of  $X_t$  will be concentrated mostly around the end-point where the extrapolation procedure is still reasonably accurate. Thus, by using the method developed here, one can construct an arbitrarily accurate representation of the solution in the *entire interpolation interval* by simply choosing a sufficiently large number of interpolation nodes and a sufficiently small time step. In general, this is not true for the finite difference method. In order to illustrate this phenomenon, we will now repeat the above calculation with a time step, which is three times as small (accordingly, the calculation will run about three times longer).

```
In[36]:= mm = U[60, 3, 60, 40, 38, 20] // Timing; mm[[1]]
```

```
Out[36]= 14.89 Second
```

```
In[37]:= Plot[{Lambda[x], ListInterpolation[mm[[2]], {{40 - 38, 40 + 38}}][x]},
  {x, 0, 80}, PlotPoints -> 200, PlotStyle -> {Hue[0.7], Hue[0.]}]
```



Let us remember that here we are also solving an optimal stopping problem, whose solution is given by the point, which separates the “exercise” region, that is, the region where the price of the option coincides with the termination payoff, from the “hold” region, where the price of the option is larger than the termination payoff.

```
In[38]:= FindRoot[ListInterpolation[mm[2], {{40 - 38, 40 + 38}}][x] == Max[40 - x, 0],
             {x, 38, 40}]
```

```
Out[38]:= {x -> 34.3}
```

It is interesting that the solution, which was produced earlier and about three times faster by using a three times larger time step, gives the same result.

```
In[39]:= FindRoot[ListInterpolation[m[2], {{40 - 38, 40 + 38}}][x] == Max[40 - x, 0],
             {x, 38, 40}]
```

```
Out[39]:= {x -> 34.3}
```

This illustrates perfectly the point that we made earlier: the result produced with the larger time step was already reasonably accurate in the middle of the interpolation interval.

Now we will compute the price of the same put option, but under the assumption that the stock price exhibits stochastic volatility of the type we introduced earlier in conjunction with European-type options.

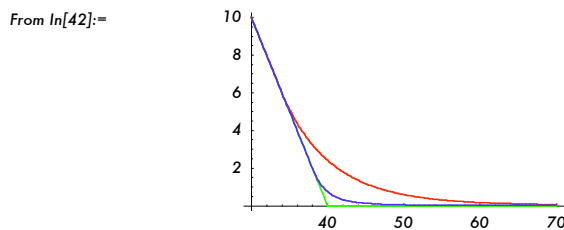
```
In[40]:=  $\sigma[x_] := 0.2 \times (1 - 0.5 e^{-0.01 \times (40-x)^2}) \times x;$ 
            $r = 0.1; a[x_] := r \times x; \Lambda[x_] := \text{Max}[40 - x, 0]$ 
```

```
In[41]:= mx = U[60, 3, 60, 40, 38, 30] // Timing; mx[1]
```

```
Out[41]:= 26.547 Second
```

Now we can compare the last result (the blue curve below) with the one that was produced in the case of constant volatility (the red curve).

```
In[42]:= Plot[{ $\Lambda[x]$ , ListInterpolation[mm[2], {{40 - 38, 40 + 38}}][x],
              ListInterpolation[mx[2], {{40 - 38, 40 + 38}}][x]}, {x, 30, 70},
             PlotPoints -> 200, PlotStyle -> {Hue[0.3], Hue[0.], Hue[0.7]}]
```



We can see that, just as one would expect, the price of this option differs from the one produced under the constant volatility assumption only in the region where the volatility decreases. Also, the decrease of the volatility near the exercise price actually decreases the range of stock prices for which immediate exercise is optimal, as the following computation shows.

```
In[43]:= FindRoot[ListInterpolation[mx[2], {{40 - 38, 40 + 38}}][x] == Max[40 - x, 0],
             {x, 23, 30}]
```

```
Out[43]:= {x -> 30.}
```

## ■ 7. Concluding Remarks

Although the examples presented in Section 6 were borrowed mostly from the realm of computational finance, it is important to realize that neither the general method developed in Sections 3 and 5, nor the *Mathematica* code used in Section 6 takes advantage of the special nature of the problems associated with financial applications in any way. The same method—and, indeed, the same computer code—can produce an approximate solution to a rather general one-dimensional parabolic PDE with boundary conditions imposed on some free or fixed boundary. Furthermore, this method does not require any manipulation whatsoever of the PDE.

As is well known, the “no free lunch” principle reigns in the world of computing, too, and the simplicity, the generality, and the precision with which we were able to solve some classical option valuation problems in the previous section did not come easily by any means. In terms of this metaphor, not only was the lunch not free, but it was, in fact, a rather expensive one. Indeed, an efficient integration procedure was absolutely crucial for the implementation of the method in Section 6. As explained in *The Mathematica Book* (see [8] p. 1071) the function `Integrate[]` “uses about 500 pages of *Mathematica* code and 600 pages of C code.” The role played by the function `ListInterpolation[]` was just as crucial. Indeed, it allowed us to work on an exceptionally coarse grid in the state-space and then “restore” the values at all remaining points by a fairly sophisticated interpolation procedure. More importantly, we were able to feed `NIntegrate[]` with the object produced by `ListInterpolation[]`. There are intrinsic obstacles to such “shortcuts” in essentially all modifications of the finite difference method.

## ■ Acknowledgment

This work is dedicated to D. W. Stroock and S. R. S. Varadhan. All numerical procedures used here are a mere transcript of their insights.

## ■ References

- [1] F. Black and M. Scholes, “The Pricing of Options and Corporate Liabilities,” *Journal of Political Economy*, **81**, 1973 pp. 637–659.
- [2] D. W. Stroock and S. R. S. Varadhan, “Multidimensional Diffusion Processes,” *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*, **233**, New York, Berlin: Springer-Verlag, 1979.
- [3] D. W. Stroock, “Lectures on Stochastic Analysis: Diffusion Theory,” *London Mathematical Society Student Texts*, **6**, 1987, Oxford: Cambridge University Press.
- [4] D. W. Stroock, *Probability Theory, an Analytic View*, Oxford: Cambridge University Press, 1993.
- [5] A. Lyasoff, “Three Page Primer in Financial Calculus.” (2000) Boston: Boston University. [www.bu.edu/mathfn/people/primer1.pdf](http://www.bu.edu/mathfn/people/primer1.pdf).

- [6] P. E. Kloeden and E. Platen, *Numerical Solutions of Stochastic Differential Equations*, New York: Springer-Verlag, 1997.
- [7] D. Stroock and S. Taniguchi, "Diffusions as Integral Curves, or Stratonovich without Itô," in *Progress in Probability #34*, ed. by M. Freidlin, Birkhäuser, 1994, pp. 331–369.
- [8] S. Wolfram, *The Mathematica Book*, 5th ed. Champaign, Oxford: Wolfram Media/Cambridge University Press, 1999.

## About the Author

Andrew Lyasoff is associate professor and director of the M.A. degree program in Mathematical Finance at Boston University. His research interests are mostly in the area of Stochastic Calculus and Mathematical Finance. Over the last few years he has been using *Mathematica* extensively to develop new numerical methods for PDEs and new instructional tools for graduate courses in Operations Research and Mathematical Finance. In addition, he is developing new theoretical models that reflect long-range dependencies in market data.

### Andrew Lyasoff

*Boston University  
Department of Mathematics and Statistics  
Boston, MA  
alyasoff@bu.edu*