Dynamic Integration of Interpolating Functions and Some Concrete Optimal Stopping Problems

Andrew Lyasoff

This article describes a streamlined method for simultaneous integration of an entire family of interpolating functions that uses one and the same interpolation grid in one or more dimensions. A method for creating customized quadrature/cubature rules that takes advantage of certain special features of *Mathematica*'s `InterpolatingFunction` objects is presented. The use of such rules leads to a new and more efficient implementation of the method for optimal stopping of stochastic systems that was developed in [1]. In particular, this new implementation allows one to extend the scope of the method to free boundary optimal stopping problems in higher dimensions. Concrete applications to finance—mainly to American-style financial derivatives—are presented. In particular, the price of an American put option that can be exercised with any one of two uncorrelated underlying assets is calculated as a function of the observed prices. This method is similar in nature to the well-known Longstaff-Schwartz algorithm, but does not involve Monte-Carlo simulation of any kind.

■ Preliminaries

The most common encounter with the concept of dynamic programming—and here we will be concerned only with continuous time dynamic programming (CTDP)—occurs in the context of the optimal stopping of an observable stochastic process with given termination payoff. This is a special case of stochastic optimal control where the control consists of a simple on-off switch, which, once turned off, cannot be turned on again. A generic example of an optimal stopping problem can be described as this: the investor observes a continuous Markov process $(X_t)_{t \geq 0}$ with state-space $\mathfrak{S} \subseteq \mathbb{R}^N$ and with known stochastic

dynamics and, having observed at time $t \geq 0$ the value $\hat{X}_t \in \mathfrak{S}$, where $\hat{X}_t$ denotes the observed realization of the random variable X_t, the investor must decide whether to terminate the process immediately and collect the payoff $\Lambda(\hat{X}_t)$, where $\Lambda : \mathfrak{S} \mapsto \mathbb{R}$ is some a priori determined (deterministic) payoff function, or to wait until time $t + \Delta > t$ for the next opportunity to terminate the process (in the context of CTDP, the time step Δ should be understood as an infinitesimally small quantity). In addition, having observed the quantity $\hat{X}_t \in \mathfrak{S}$, the investor must determine the value of the observable system (associated with that state) at time $t \geq 0$. In many situations, the stopping of the process cannot occur after some finite deterministic time $T > 0$.

There is a vast literature that deals with both the theoretical and the computational aspects of modeling, analysis, and optimal control of stochastic systems. The works of Bensoussan and Lions [2] and Davis [3] are good examples of classical treatments of this subject.

Many important problems from the realm of finance can be formulated—and solved—as optimal stopping problems. For example, a particular business can be established at the fixed cost of $\mathcal{I}$ dollars, but the actual market price of that business follows some continuous stochastic process $(X_t)_{t \geq 0}$, which is observable. Having observed at time t the value $\hat{X}_t$, an investor who owns the right to incorporate such a business must decide whether to exercise that right immediately, in which case the investor would collect the payoff of $\hat{X}_t - \mathcal{I}$ dollars, or to postpone the investment decision until time $t + \Delta$, with the hope that the business will become more valuable. In addition, having observed the value $\hat{X}_t$, the investor may need to determine the market price of the guaranteed right (say, a patent) to eventually incorporate this business—now, or at any time in the future.

Another classical problem from the realm of finance is the optimal exercise of a stock option of American type, that is, an option that can be exercised at any time prior to the expiration date. A stock option is a contract that guarantees the right to buy (a *call option*) or to sell (a *put option*) a particular stock (the *underlying*) at some fixed price (the *strike price*, stated explicitly in the option contract) at any time prior to the expiration date T (the *maturity date*, also stated in the option contract). Consider, for example, an American style put option which expires at some future date $T > 0$. On date $t < T$, the holder of the option observes the underlying price $\hat{X}_t$ and decides whether to exercise the option, that is, to sell the stock at the guaranteed price K (and, consequently, collect an immediate payoff of $K - \hat{X}_t$ dollars) or to wait, hoping that at some future moment, but no later than the expiration date T, the price will fall below the current level $\hat{X}_t$. If the option is not exercised before the expiration date, the option is lost if $X_T \geq K$ or, ignoring any transaction costs, is always exercised if $X_T < K$; that is, if the option is not exercised prior to the expiration date T, on that date the owner of the option collects the amount $\text{Max}[K - X_T, 0]$. In general, stock options are traded in the same way that stocks are traded and, therefore, have a market value determined by the laws of supply and demand. At the same time, the price of such

contracts, treated as a function of the observed stock price $\hat{X}_t$, can be calculated from general economic principles in conjunction with the principles of dynamic programming.

In general, the solution to any optimal stopping problem has two components: (1) for every moment in time, t, one must obtain a termination rule in the form of a termination set $\mathcal{E}_t \subseteq \mathfrak{S}$ so that the process is terminated at the first moment t at which the event $\{X_t \in \mathcal{E}_t\}$ occurs; and (2) for every moment t, one must determine the value function $f_t : \mathfrak{S} \mapsto \mathbb{R}$, which measures how "valuable" different possible observations are at time t. Clearly, if $\Lambda(\cdot)$ denotes the termination payoff function (e.g., in the case of an American style put option that would be $\Lambda(x) = \text{Max}[K - x, 0]$, $x \in \mathbb{R}_+$), then for every $x \in \mathcal{E}_t$ one must have $f_t(x) = \Lambda(x)$, while for every $x \notin \mathcal{E}_t$ one must have $f_t(x) > \Lambda(x)$; that is, one would choose to continue only when continuation is more valuable than immediate termination. Consequently, the termination sets $\mathcal{E}_t$, $t \geq 0$, can be identified with the sets $\left\{x \in \mathbb{R}^N;\ f_t(x) = \Lambda(x)\right\}$, and the associated optimal stopping time can be described as

$$\tau = \inf\{t \in [0,\ T];\ f_t(X_t) = \Lambda(X_t)\}.$$

Thus, the entire solution to the optimal stopping problem can be expressed in terms of the family of value functions $\{f_t(\cdot);\ t \in [0,\ T]\}$, which may be treated as a single function of the form

$$[0,\ T] \times \mathfrak{S} \ni (t,\ x) \longrightarrow f_t(x) \in \mathbb{R}.$$

It is important to recognize that the solution to the optimal stopping problem, that is, the value function $(t,\ x) \longrightarrow f_t(x)$, must be computed before the observation process has begun.

In most practical situations, an approximate solution to the optimal stopping problem associated with some observable process $(X_t)_{t \geq 0}$ and some fixed termination payoff $\Lambda : \mathfrak{S} \mapsto \mathbb{R}$ may be obtained as a finite sequence of approximate value functions

$$\langle f \rangle_t^{\nu} : \mathfrak{S} \mapsto \mathbb{R}, \qquad t = T,\ T - \frac{T}{\nu},\ T - 2\,\frac{T}{\nu},\ \ldots,\ 0,$$

where $T > 0$ is the termination date and ν is some sufficiently large integer number. This approximate solution can be calculated from the following recursive rule, which is nothing but a special discrete version of what is known as the *dynamic programming equation*: for $t = T$, we set $\langle f \rangle_t^{\nu}(\cdot) \equiv \Lambda(\cdot)$, that is, at time $t = T$, the value function coincides with the termination payoff on the entire state-space $\mathfrak{S}$, and for $t = T - \frac{T}{\nu}, T - 2\,\frac{T}{\nu}, \ldots$, we set

$$\langle f \rangle_t^{\nu}(x) := \text{Max}\left[\Lambda(x),\ e^{-r_t\,T/\nu}\,\text{E}_Q\left[\langle f \rangle_{t+T/\nu}^{\nu}\left(X_{t+T/\nu}\right) \;\middle|\; X_t = x\right]\right], \qquad x \in \mathfrak{S}, \tag{1}$$

where r_t is the instantaneous discount rate at time t, and the probability measure Q, with respect to which the conditional expectation is calculated, is the so-called *pricing measure* (in general, the instantaneous discount rate $r_t \equiv r_t(x)$ may depend on the observed position $x = \hat{X}_t$ and the pricing measure may be different from

the probability measure that governs the actual stochastic dynamics of the process $(X_t)_{t\geq 0}$). Furthermore, for every $t = T - \frac{T}{\nu}, T - 2\frac{T}{\nu}, \ldots$, the approximate termination set is given by

$$\langle \mathcal{E} \rangle_t \equiv \left\{ x \in \mathbb{R}^N ; \langle f \rangle_t^{\nu}(x) = \Lambda(x) \right\}.$$

If, given any $t \in [0, T]$, the limit

$$f_t(\cdot) \equiv \lim_{\nu \nearrow \infty} \langle f \rangle^{\nu}_{(T/\nu)\lfloor \nu t/T \rfloor}(\cdot) \tag{2}$$

($\lfloor \cdot \rfloor$ denotes the integer part of a real number) exists, in some appropriate topology on the space of functions defined on the state-space $\mathfrak{S}$, then one can declare that the function

$$[0, T] \times \mathbb{R}^N \ni (t, x) \longrightarrow f_t(x) \in \mathbb{R}$$

gives *the* solution to the optimal stopping problem with observable process $(X_t)_{t\geq 0}$, payoff function $\Lambda(\cdot)$, *and* pricing measure Q.

It is common to assume that under the pricing probability measure Q the stochastic dynamics of the observable process $(X_t)_{t\geq 0}$ are given by some diffusion equation of the form

$$d X_t = \sigma(X_t)\, d W_t + b(X_t)\, d t \tag{3}$$

for some (sufficiently nice) matrix-valued function $\mathbb{R}^N \ni x \longrightarrow \sigma(x) \in \mathbb{R}^{N\otimes N}$, vector field $\mathbb{R}^N \ni x \longrightarrow b(x) \in \mathbb{R}^N$, and some $\mathbb{R}^N$-valued Brownian motion process $(W_t)_{t\geq 0}$, which is independent from the starting position X_0 (note that $(W_t)_{t\geq 0}$ is Brownian motion with respect to the law Q). Given any $x \in \mathbb{R}^N$, let $A(x)$ denote the matrix $\sigma(x)\,\sigma(x)^T$ and let $\mathcal{L}$ denote the second-order field in $\mathbb{R}^N$ given by

$$\mathbb{R}^N \ni x \longrightarrow \mathcal{L}_x \equiv \frac{1}{2} \sum_{i,j=1}^{N} A(x)_{i,j} \frac{\partial^2}{\partial_{x_i} \partial_{x_j}} + \sum_{i=1}^{N} b(x)_i \frac{\partial}{\partial_{x_i}}.$$

As is well known, under certain fairly general conditions for the coefficients $\sigma(\cdot)$ and $b(\cdot)$, and the termination payoff $\Lambda : \mathfrak{S} \mapsto \mathbb{R}$, the value function

$$[0, T] \times \mathfrak{S} \ni (t, x) \longrightarrow f_t(x) \in \mathbb{R},$$

that is, the solution to the associated optimal stopping problem, is known to satisfy the following equation, which is nothing but a special case of the Hamilton-Jacobi-Bellman (HJB) equation

$$r_t(x)\, f_t(x) = \mathrm{Max}\left[r_t(x)\, \Lambda(x),\ \partial_t f_t(x) + \mathcal{L}_\xi f_t(x) \right], \qquad x \in \mathbb{R}^N, \tag{4}$$

with boundary condition $f_T(x) = \Lambda(x)$, $x \in \mathbb{R}^N$. This equation is a more or less trivial consequence of equations (1) and (2), in conjunction with the Itô formula. It is important to recognize that the dynamic programming equation (1) is primary for the optimal stopping problem, while the HJB equation (4) is only secondary, in that it is nothing more than a computational tool. Unfortunately, equation (4) admits a closed form solution only for some rather special choices of the payoff function $\Lambda(\cdot)$, the diffusion coefficients $b(\cdot)$ and $\sigma(\cdot)$, and the discount rate $r_t(\cdot)$. Thus, in most practical situations, the HJB equation (4)

happens to be useful only to the extent to which this equation gives rise to some numerical procedure that may allow one to compute the solution $(t, x) \longrightarrow f_t(x)$ approximately. Essentially, all known numerical methods for solving equation (4) are some variations of the finite difference method. The most common approach is to reformulate equation (4) as a *free boundary value problem*: one must find a closed domain $\mathcal{D} \subseteq [0, T[\times \mathfrak{S}$ with a piecewise differentiable boundary $\partial\mathcal{D}$ and nonempty interior $\mathcal{D}^\circ \equiv \mathcal{D} \setminus \partial\mathcal{D}$, plus a continuous function $u : \mathcal{D} \mapsto \mathbb{R}$, which is continuously differentiable with respect to the variable $t \in]0, T[$ and is twice continuously differentiable with respect to the variables $x \in \mathcal{D}^\circ$, that satisfies the following two relations everywhere inside $\mathcal{D}^\circ$

$$r_t(x)\, u(t, x) = \partial_t u(t, x) + \mathcal{L}_x u(t, x) \quad \text{and} \quad u(t, x) > \Lambda(x), \qquad (t, x) \in \mathcal{D}^\circ,$$

and, finally, satisfies the following boundary conditions along the free (i.e., unknown) boundary $\partial\mathcal{D}$

$$u(t, x) = \Lambda(x) \quad \text{and} \quad \frac{\partial}{\partial_{x_i}} u(t, x) = \frac{\partial}{\partial_{x_i}} \Lambda(x), \qquad 1 \le i \le n, \qquad (t, x) \in \partial\mathcal{D}.$$

The last two conditions are known, respectively, as the *value matching* and the *smooth pasting* conditions. It can be shown that under some rather general—but still quite technical—assumptions, the free boundary value problem that was just described has a unique solution (consisting of a function $u(\cdot, \cdot)$ and domain $\mathcal{D}$) which then allows one to write the solution to the optimal stopping problem as follows: for any $t \in [0, T]$ and any $x \in \mathfrak{S}$, one has

$$f_t(x) = \begin{cases} u(t, x), & \text{for } (t, x) \in \mathcal{D}; \\ \Lambda(x), & \text{for } (t, x) \in \mathcal{D}^c \equiv ([0, T] \times \mathfrak{S}) \setminus \mathcal{D}. \end{cases}$$

The drawbacks from formulating an optimal stopping problem as a free boundary value problem for some parabolic partial differential equation (PDE), which—in principle, at least—can be solved numerically by way of finite differencing, are well known. First, the value matching and the smooth pasting conditions are difficult to justify and this makes the very formulation of the problem rather problematic in many situations. Second, just in general, for purely technical reasons, it is virtually impossible to use finite differencing when the dimension of the state-space is higher than 3 and, in fact, in the case of free boundary value problems, even a state-space of dimension 2 is rather challenging. Third, with the exception of the explicit finite difference scheme, which is guaranteed to be stable only under some rather restrictive conditions, the implementation of most finite differencing procedures on parallel processors is anything but trivial.

At the time of this writing, it is safe to say that finite differencing is no longer at the center of attention in computational finance, where most problems are inherently complex and multidimensional. Indeed, most of the research in computational finance in the last five years or so appears to be focused on developing new simulation-based tools—see [4-8], for example (even a cursory review of the existing literature from the last few years is certain to be very long and is beyond the scope of this article). Among these methods, the Longstaff-Schwartz algorithm [8] seems to be the most popular among practitioners.

The attempts to avoid the use of finite differencing are nothing new (there is a section in the book *Numerical Recipes in* C entitled There Is More to Life than Finite Differencing—see [9], p. 833). Indeed, Monte Carlo, finite element, and several other computational tools have been in use for solving fixed boundary value problems for PDEs for quite some time. Apparently, from the point of view of stochastic optimal control, it is more efficient—and in some ways more natural—to develop numerical procedures directly from the dynamic programming equation (1) and skip the formulation of the HJB equation altogether—at least this is the approach that most recent studies in computational finance are taking.

Since the solution to an optimal stopping problem is given by an infinite family of functions of the form $f_t : \mathfrak{S} \mapsto \mathbb{R}$, $t \in [0, T]$, for some set $\mathfrak{S} \subseteq \mathbb{R}^N$, from a computational point of view the actual representation of any such object involves two levels of discretization. First, one must discretize the state-space $\mathfrak{S}$, that is, functions defined on $\mathfrak{S}$ must be replaced with finite lists of values attached to some fixed set of distinct abscissas

$$\{x_k \in \mathfrak{S};\ 1 \le k \le n\},$$

for some fixed $n \in \mathbb{Z}_+$. This means that for every list of values $\ell \in \mathbb{R}^n$, one must be able to construct a function $S^\ell : \mathfrak{S} \mapsto \mathbb{R}$ which "extends" the discrete assignment $x_k \longrightarrow \ell_k$, $1 \le k \le n$, to some function defined on the entire state-space $\mathfrak{S}$. Second, one must discretize time, that is, replace the infinite family of functions $\{f_t(\cdot);\ t \in [0, T]\}$ not just with the finite sequence of functions

$$\langle f \rangle_t^\nu : \mathfrak{S} \mapsto \mathbb{R}, \qquad t = T,\ T - \frac{T}{\nu},\ T - 2\,\frac{T}{\nu},\ \ldots,\ 0,$$

determined by the discrete dynamic programming equation (1), but, rather, with a finite sequence of lists

$$\ell(t) \in \mathbb{R}^n, \qquad t = T - \frac{T}{\nu},\ T - 2\,\frac{T}{\nu},\ \ldots,\ 0,$$

computed from the following space-discretized analog of the time-discrete dynamic programming equation (1): for $t = T - \frac{T}{v}$, we set

$$\ell(t)_k \equiv \mathrm{Max}\big[\Lambda(x_k),\ e^{-r_t\,(T/\nu)}\,\mathrm{E}_Q\big[\Lambda(X_T) \mid X_t = x_k\big]\big], \qquad 1 \le k \le n,$$

and then define

$$\ell(t)_k \equiv \mathrm{Max}\Big[\Lambda(x_k),\ e^{-r_t(T/\nu)}\,\mathrm{E}_Q\Big[S^{\ell(t+T/\nu)}\big(X_{t+T/\nu}\big)\ \Big|\ X_t = x_k\Big]\Big], \qquad 1 \le k \le n, \tag{5}$$

consecutively, for $t = T - 2\,\frac{T}{v}$, $t = T - 3\,\frac{T}{v}$, $\ldots$, where $S^{\ell(t+T/\nu)} : \mathfrak{S} \mapsto \mathbb{R}$ is simply the map that "extends" the discrete assignment $x_k \longrightarrow \ell\big(t + \frac{T}{v}\big)_k$.

Of course, in order to compute the conditional expectation in equation (5), the distribution law of the random variable $X_{t+T/\nu}$ relative to the pricing measure Q and conditioned to the event $\{X_t = x_k\}$ must be expressed in computable form.

Unfortunately, this is possible only for some rather special choices for the diffusion coefficients $\sigma(\cdot)$ and $b(\cdot)$ that govern the stochastic dynamics of the process $(X_t)_{t \geq 0}$ under the pricing measure Q. This imposes yet another—third—level of approximation. The simplest such approximation is to replace $X_{t+T/\nu}$ in equation (5) with the random quantity

$$\psi_{T/\nu}(x_k, \Gamma) = x_k + \sigma(x_k) \sqrt{\frac{T}{\nu}}\ \Gamma + b(x_k)\left(\frac{T}{\nu}\right), \tag{6}$$

where Γ is some standard normal $\mathbb{R}^N$-valued random variable. As a result of this approximation, expression (5) becomes

$$\ell(t)_k \equiv \text{Max}\left[\Lambda(x_k),\ e^{-r_t (T/\nu)} \int_{\mathbb{R}^N} S^{\ell(t+T/\nu)}\left(\psi_{T/\nu}(x_k, x)\right) p_N(x)\, dx\right], \quad 1 \leq k \leq n, \tag{7}$$

where $p_N(\cdot)$ stands for the standard normal probability density function in $\mathbb{R}^N$. Another (somewhat more sophisticated) approximation scheme for the diffusion process $(X_t)_{t \geq 0}$ in the case $N = 1$ is discussed in [1]. Fortunately, in the most widely used diffusion models in finance, the conditional law of $X_{t+T/\nu}$, given the event $\{X_t = x_k\}$, coincides with the law of a random variable of the form $\psi(x_k, \Gamma)$ for some function $\psi(x_k, \cdot)$ that can be expressed in computable form, so that in this special case the right side of equation (7) is exactly identical to the right side of equation (5) and the approximation given in equation (6) is no longer necessary.

One must be aware that, in general, the distribution law of the random variable $\psi(x_k, \Gamma)$ is spread on both sides of the point x_k and, even though for a reasonably small time step $\frac{T}{\nu}$ this law is concentrated almost exclusively in some small neighborhood of the node x_k, when this node happens to be very close to the border of the interpolation region, the computation of the integral on the right side of equation (7) inevitably requires information about the function $S^{\ell(t+T/\nu)}(\cdot)$ outside the interpolation domain. However, strictly speaking, interpolating functions are well defined only inside the convex hull of the abscissas used in the interpolation. One way around this problem is to choose the interpolation domain in such a way that along its border the solution to the optimal stopping problem takes values that are very close to certain asymptotic values. For example, if one is pricing an American put option with strike price of \$1, one may assume that the value of the option is practically 0 when the price of the underlying asset is above \$100, regardless of the time left to maturity (of course, this assumption may be grossly inaccurate when the volatility is very large and/or when the option matures in the very distant future). Consequently, for those nodes x_k that are close to the border of the interpolation domain, the respective values $\ell(t)_k$ will not depend on t and will not be updated according to the prescription (7). At the same time, the remaining nodes x_k must be chosen in such a

way that the probability mass of the random variable $\psi(x_k, \Gamma)$ is concentrated almost exclusively inside the interpolation region.

In general, numerical procedures for optimal stopping of stochastic systems differ in the following: First, they differ in the space-discretization method, that is, the method for choosing the abscissas x_k and for "reconstructing" the function $S^{\ell}(\cdot)$ from the (finite) list of values ℓ assigned to the abscissas x_k. Second, numerical procedures differ in the concrete quadrature rule used in the calculation of the conditional expectation in the discretized dynamic programming equation (5). The method described in this article is essentially a refinement of the method of dynamic interpolation and integration described in [1]. The essence of this method is that the functions $S^{\ell}(\cdot)$ are defined by way of spline interpolation—or some other type of interpolation—from the list of values ℓ, while the integral in equation (7) is calculated by using some standard procedures for numerical integration. As was illustrated in [1], the implementation of this procedure in *Mathematica* is particularly straightforward, since `NIntegrate` accepts as an input `InterpolatingFunction` objects, and, in fact, can handle such objects rather efficiently.

The key point in the method developed in this article is a procedure for computing a universal list of vectors $\Omega_k \in \mathbb{R}^n$, $1 \le k \le n$, associated with the abscissas x_k, $1 \le k \le n$, that depend only on the stochastic dynamics of $(X_t)_{t \ge 0}$ and the time step $\frac{T}{v}$, so that one can write

$$\int_{\mathcal{R}} S^{\ell(t+T/v)}\big(\psi(x_k, x)\big)\, p_N(x)\, dx = \Omega_k \cdot \ell\left(t + \frac{T}{v}\right) \tag{8}$$

for any t and for any k, where $\mathcal{R}$ stands for the convex hull of the interpolation grid x_k, that is, the domain of interpolation. Essentially, equation (8) is a variation of the well-known cubic spline quadrature rule—see §4.0 in [9] and §5.2, p. 89, in [10]. It is also analogous to the "cubature formula," which was developed in [11] with the help of a completely different computational tool (following [14] and [11], we use the term *cubature* as a reference to some integration rule for multiple integrals and the term *quadrature* as a reference to some inte gration rule for single integrals). One of the principal differences between the interpolation quadrature/cubature rules and the cubature formula obtained in [11] is that the former allows for a greater freedom in the choice of the abscissas x_k. This means that, in practice, one can choose the evaluation points in a way that takes into account the geometry of the payoff function $\Lambda(\cdot)$ and not just the geometry of the diffusion $(X_t)_{t \ge 0}$. From a practical point of view, this feature is quite important and, in fact, was the main reason for developing what is now known as *Gaussian quadratures*.

An entirely different method for computing conditional averages of the form

$$\mathrm{E}_Q\Big[\varphi\big(X_{t+T/v}\big) \;\Big|\; X_t = x_k\Big]$$

was developed in [12] and [13]. In this method, quantities of this form are approximated by finite products of linear operators acting on the function $\varphi(\cdot)$. One

must be aware that the methods described in [11-13] require a certain degree of smoothness for the integrand $\varphi(\cdot)$, which, generally, `InterpolatingFunction` objects may not have.

The method developed in this article bears some similarity also with the quantization algorithm described in [4, 5]. The key feature in both methods is that one can separate and compute—once and for all—certain quantities that depend on the observable process but not on the payoff function. Essentially, this means that one must solve, simultaneously, an entire family of boundary value problems corresponding to different payoff functions. The two methods differ in the way in which functions on the state-space are encoded in terms of finite lists of values. The advantages in using interpolating functions in general, or splines in particular, are well known: one can "restore" the pricing maps from fewer evaluation points and this feature is crucial for optimal stopping problems in higher dimensions.

■ Making Quadrature/Cubature Rules with *Mathematica*

In this section we show how to make quadrature/cubature rules for a fixed set of abscissas that are similar to the well-known Gaussian rules or the cubic spline rules. We consider quadrature rules on the real line $\mathbb{R}$ first. To begin with, suppose that one must integrate some cubic spline $x \longrightarrow S(x)$ that is defined from the abscissas $\{x_1, \ldots, x_n\} \in \mathbb{R}^n$, $x_1 < \cdots < x_n$, and from some list of tabulated values $\{v_1, \ldots, v_n\} \in \mathbb{R}^n$. Since $S(\cdot)$ is a cubic spline, for any fixed $1 \le k \le (n-1)$ there are real numbers $a_{k,0}$, $a_{k,1}$, $a_{k,2}$, and $a_{k,3}$ for which we have

$$S(x) = a_{k,0} + a_{k,1}\, x + a_{k,2}\, x^2 + a_{k,3}\, x^3 \quad \text{for any} \quad x \in \left[x_k,\, x_{k+1}\right].$$

As a result, one can write

$$\int_{x_1}^{x_n} S(x)\, dx = \sum_{k=1}^{n-1} \sum_{i=0}^{3} \frac{a_{k,i}}{i+1} \left(x_{k+1}^{i+1} - x_k^{i+1}\right), \tag{9}$$

which reduces the computation of the integral $\int_{x_1}^{x_n} S(\xi)\, d\xi$ to the computation of the coefficients $a_{k,i}$ from the tabulated values v_k and the abscissas x_k.

Though not immediately obvious, it so happens that when all abscissas $x_1 < \cdots < x_n$ are fixed, each coefficient $a_{k,i}$—and, consequently, the entire expression on the right side of equation (9)—can be treated as a linear function of the list $\{v_1, \ldots, v_n\} \in \mathbb{R}^n$ (in fact, each $a_{k,i}$ is a linear function only of a small portion of that list). This property is instrumental for the quadrature/cubature rules that will be developed in the next section. To see why one can make this claim, recall that (e.g., see §3.3 in [9]) the cubic spline created from the discrete assignment $x_k \longrightarrow v_k$, $1 \le k \le n$, is given by

$$S(x) \equiv a_{k,0} + a_{k,1}\, x + a_{k,2}\, x^2 + a_{k,3}\, x^3$$
$$= A_k(x)\, v_k + B_k(x)\, v_{k+1} + C_k(x)\, v''_k + D_k(x)\, v''_{k+1}, \qquad (10)$$
$$x_k \le x \le x_{k+1}, \qquad 1 \le k \le n-1,$$

where

$$A_k(x) \equiv \frac{x_{k+1} - x}{x_{k+1} - x_k},$$
$$B_k(x) \equiv 1 - A_k(x),$$
$$C_k(x) \equiv \frac{A_k(x)^3 - A_k(x)}{6} \left(x_{k+1} - x_k\right)^2,$$
$$D_k(x) \equiv \frac{B_k(x)^3 - B_k(x)}{6} \left(x_{k+1} - x_k\right)^2,$$

and $v''_k := S''\left(x_k\right)$, $1 \le k \le n$ (note that the definition implies $S\left(x_k\right) = v_k$, $1 \le k \le n$). Usually, one sets $v''_1 = v''_n = 0$ (which gives the so-called *natural splines*) and determines the remaining values $\{v''_2, \ldots, v''_{n-1}\}$ from the requirement that the cubic spline has a continuous first derivative on the entire interval $]x_1, x_n[$ (it is a trivial matter to check that with this choice the second derivative of the function defined in equation (10) is automatically continuous on $]x_1, x_n[$). Finally, recall that this last requirement leads to a system of $n-2$ linear equations with unknowns $\{v''_1, \ldots, v''_n\}$ and that the right side of each of these equations is a linear function of the list $\{v_1, \ldots, v_n\}$, while the coefficients in the left side are linear functions of the abscissas $\{x_1, \ldots, x_n\}$—see equation (3.3.7) in [9], for example. This means that each quantity in the list $\{v''_2, \ldots, v''_{n-1}\}$ can be treated as a linear function of the list $\{v_1, \ldots, v_n\}$. Since the quantities $A_k(x)$, $B_k(x)$, $C_k(x)$, and $D_k(x)$ are all independent from the choice of $\{v_1, \ldots, v_n\}$, one can claim that the entire expression in the right side of equation (10) is a linear function of the list $\{v_1, \ldots, v_n\}$. Thus, for every $1 \le k \le (n-1)$ and every $x \in [x_i, x_{i+1}]$, we can write

$$S(x) \equiv a_{k,0} + a_{k,1}\, x + a_{k,2}\, x^2 + a_{k,3}\, x^3 = \sum_{k=1}^{n} h_{k,i}(x)\, v_k\,,$$

where $h_{k,i}(\cdot)$ are polynomials of degree at most 3. More importantly, these polynomials are universal in the sense that they depend on the abscissas x_k but not on the tabulated values v_k. In fact, this representation holds for spline interpolation objects of any degree, except that in the general case the degree of the polynomials $h_{k,i}(\cdot)$ may be bigger than 3. An analogous representation is valid for other (non-spline) interpolating functions $S(\cdot)$; in particular, such a representation is valid for interpolating functions $S(\cdot)$ defined by way of divided differences interpolation, which is the method used by `ListInterpolation`. For example, if $S(\cdot)$ denotes the usual interpolating polynomial of degree $n-1$ defined from the discrete assignment $x_k \longrightarrow v_k$, $1 \le k \le n$, then $h_{k,i}(\cdot) \equiv h_k(\cdot)$ is simply the k^{th} Lagrange polynomial for the abscissas x_k. Thus, when $S(\cdot)$ is an interpolating function object (defined by, say, splines, divided differences, or standard polyno-

mial interpolation) from the assignment $x_k \longrightarrow v_k$, $1 \le k \le n$, assuming that $\phi(\cdot)$ is some (fixed) continuous and strictly monotone function and that $p(\cdot)$ is some (fixed) integrable function and, finally, assuming that the values $z_k \in \mathbb{R}$, $1 \le k \le n$, are chosen so that $\phi(z_k) = x_k$, $1 \le k \le n$, one can write

$$\int_{z_1}^{z_n} S(\phi(x))\, p(x)\, dx = \{\omega_1 \ldots, \omega_n\} \cdot \{v_1, \ldots, v_n\} \equiv \sum_{k=1}^{n} \omega_k\, v_k, \tag{11}$$

where

$$\omega_k \equiv \sum_{i=1}^{N-1} \int_{z_i}^{z_{i+1}} h_{k,i}(\phi(x))\, p(x)\, dx, \qquad 1 \le k \le n.$$

Since the list $\{\omega_1, \ldots, \omega_n\} \in \mathbb{R}^n$ can be computed once and for all and independently from the interpolated values v_k, and since the calculation of the integral $\int_{z_1}^{z_n} S(\phi(x))\, p(x)\, dx$ comes down to the calculation of the dot product $\{\omega_1 \ldots, \omega_n\} \cdot \{v_1, \ldots, v_n\}$, the computation of the list $\{\omega_1, \ldots, \omega_n\} \in \mathbb{R}^n$ may be viewed as a simultaneous integration of all functions of the form $x \longrightarrow S(\phi(x))\, p(x)$ for all interpolating functions $S(\cdot)$ (of the same interpolation type) that are based on one and the same (forever fixed) set of abscissas $\{x_1, \ldots, x_n\}$. Indeed, the knowledge of the list $\{\omega_1, \ldots, \omega_n\}$ turns the expression $\int_{z_1}^{z_n} S(\phi(x))\, p(x)\, dx$ into a trivial function of the list $\{v_1, \ldots, v_n\}$, that is, a trivial function defined on the entire class of interpolating functions $S(\cdot)$. *In other words, in equation (11) we have made a quadrature (or cubature) rule.*

The preceding remarks contain nothing new or surprising in any way and we only need to find a practical way for computing the weights ω_k. Fortunately, *Mathematica* allows one to define `InterpolatingFunction` objects even if the interpolated values v_k are defined as symbols that do not have actual numerical values assigned to them. Furthermore, such objects can be treated as ordinary functions that can be evaluated, differentiated, and so on—all in symbolic form. We illustrate this powerful feature next (recall that `ListInterpolation` has default setting `InterpolationOrder→3`).

```
In[1]:= V = Array[v#1 &, {10}]

Out[1]= {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10}

In[2]:= S = ListInterpolation[V]

Out[2]= InterpolatingFunction[{{ 1   10 }}, <>]

In[3]:= ∂x,x S(x) /. {x → 3.5} // Simplify

Out[3]= 0.5 v2 − 0.5 v3 − 0.5 v4 + 0.5 v5
```

```
In[4]:= ∂x,x S(x) /. {x → 7/2} // Simplify
```

$$\text{Out[4]=}\quad \frac{1}{2}\left(v_2 - v_3 - v_4 + v_5\right)$$

Since, just by definition, the restriction of the function $x \to S(x)$ to the interval $[3, 4]$ coincides with a polynomial of degree 3, by Taylor's theorem, for any fixed $x_0 \in]\,3, 4[$, we have

$$S(x) = \sum_{i=0}^{3} \frac{1}{i!}\, S^{(i)}(x_0)\,(x - x_0)^i \quad \text{for any} \quad 3 \le x \le 4.$$

Thus, if we define

```
In[5]:= te[ff_, x0_] := Sum[1/i! * ff^(i)(x0) * (x - x0)^i, {i, 0, 3}];
```

then on the interval $[3, 4]$ we can identify the function S with the dot product

```
In[6]:= x0 = 3.5; S34[x_] = CoefficientList[te[S, x0], x].{1, x, x^2, x^3}; Clear[x0]
```

Mathematica can rearrange this expression as an ordinary polynomial:

```
In[7]:= Collect[S34(x) // Simplify, x]
```

$$\begin{aligned}\text{Out[7]=}\quad & \left(-0.166667\, v_2 + 0.5\, v_3 - 0.5\, v_4 + 0.166667\, v_5\right) x^3 + \\ & \left(2.\, v_2 - 5.5\, v_3 + 5.\, v_4 - 1.5\, v_5\right) x^2 + \\ & \left(-7.83333\, v_2 + 19.\, v_3 - 15.5\, v_4 + 4.33333\, v_5\right) x + 10.\, v_2 - 20.\, v_3 + 15.\, v_4 - 4.\, v_5\end{aligned}$$

We can also do

```
In[8]:= x0 = 7/2; SS34[x_] = CoefficientList[te[S, x0], x].{1, x, x^2, x^3}; Clear[x0]
```

```
In[9]:= Collect[SS34(x) // Simplify, x]
```

$$\begin{aligned}\text{Out[9]=}\quad & \frac{1}{6}\left(-v_2 + 3\, v_3 - 3\, v_4 + v_5\right) x^3 + \frac{1}{6}\left(12\, v_2 - 33\, v_3 + 30\, v_4 - 9\, v_5\right) x^2 + \\ & \frac{1}{6}\left(-47\, v_2 + 114\, v_3 - 93\, v_4 + 26\, v_5\right) x + \frac{1}{6}\left(60\, v_2 - 120\, v_3 + 90\, v_4 - 24\, v_5\right)\end{aligned}$$

Essentially, this is some sort of reverse engineering of the `InterpolatingFunction` object S, that is, a way to "coerce" *Mathematica* to reveal what polynomials the object S is constructed of between the interpolation points.

There are other tricks that we can use in order to reverse engineer an `InterpolatingFunction` object. For example, with the definition of S, the entire function $[3, 4] \ni x \longrightarrow S(x) \in \mathbb{R}$ can be restored from the values $\left\{S(3), S\left(\frac{10}{3}\right), S\left(\frac{11}{3}\right), S(4)\right\}$. To do the restoration we must solve a linear system for the four unknown coefficients in some (unknown) cubic polynomial. This can be imple-

mented in *Mathematica* rather elegantly without the use of `CoefficientList`. The only drawback in this approach is that one must resort to `Solve` or `Linear`⁻. `Solve`, which, generally, are more time-consuming, especially in dimensions 2 or higher. Of course, instead of using reverse engineering, one can simply implement the interpolation algorithm step-by-step, which, of course, involves a certain amount of work.

For example, we have

```
In[10]:= S34(3.7) – S(3.7) // Simplify
```

$$\text{Out[10]}= 3.55271\times 10^{-15}\, v_2 - 1.42109\times 10^{-14}\, v_3 + 0.\, v_4 - 7.10543\times 10^{-15}\, v_5$$

```
In[11]:= % // Chop
```

$$\text{Out[11]}= 0$$

```
In[12]:= SS34[37/10] – S[37/10] // Simplify
```

$$\text{Out[12]}= 0$$

In the definition of `S34`, the variable `x0` can be given any numerical value inside the interval]3, 4[; the choice `x0=3.5` was completely arbitrary.

In the same way, one can reverse engineer `InterpolatingFunction` objects that depend on any number of independent abscissas. Here is an example.

```
In[13]:= VV = Array[v#1,#2 &, {10, 10}];
```

```
In[14]:= H = ListInterpolation[VV]
```

$$\text{Out[14]}= \text{InterpolatingFunction}\left[\begin{pmatrix} 1 & 10 \\ 1 & 10 \end{pmatrix}, <>\right]$$

```
In[15]:= ∂x,y H(x, y) /. {x → 3.2, y → 3.7} // Simplify
```

$$\begin{aligned}\text{Out[15]}= {} & -0.0186556\, v_{2,2} + 0.178633\, v_{2,3} - 0.147967\, v_{2,4} - 0.0120111\, v_{2,5} - 0.1022\, v_{3,2} + \\ & 0.9786\, v_{3,3} - 0.8106\, v_{3,4} - 0.0658\, v_{3,5} + 0.1387\, v_{4,2} - 1.3281\, v_{4,3} + 1.1001\, v_{4,4} + \\ & 0.0893\, v_{4,5} - 0.0178444\, v_{5,2} + 0.170867\, v_{5,3} - 0.141533\, v_{5,4} - 0.0114889\, v_{5,5}\end{aligned}$$

```
In[16]:= Te[ff_, x0_, y0_] :=
           Sum[1/(i!)*(j!) * ff^(i,j)(x0, y0) * (x – x0)^i * (y – y0)^j, {i, 0, 3}, {j, 0, 3}];
```

```
In[17]:= x0 = 3.5; y0 = 4.5;
         H3445[x_, y_] = Flatten[CoefficientList[Te[H, x0, y0], {x, y}] // Simplify].
           Flatten[Table[x^i y^j, {i, 0, 3}, {j, 0, 3}]]; Clear[x0, y0]
```

```
In[18]:= H3445(3.2, 4.7) – H(3.2, 4.7) // Simplify
```

$$\begin{aligned}\text{Out[18]=}\ & 1.7053\times 10^{-13}\, v_{2,3} + 0.\, v_{2,4} - 4.54747\times 10^{-13}\, v_{2,5} + \\ & 1.13687\times 10^{-13}\, v_{2,6} + 0.\, v_{3,3} - 7.27596\times 10^{-12}\, v_{3,4} - \\ & 2.72848\times 10^{-12}\, v_{3,5} - 1.81899\times 10^{-12}\, v_{3,6} - 2.04636\times 10^{-12}\, v_{4,3} + \\ & 2.27374\times 10^{-13}\, v_{4,4} - 2.27374\times 10^{-12}\, v_{4,5} + 1.53477\times 10^{-12}\, v_{4,6} + \\ & 3.41061\times 10^{-13}\, v_{5,3} + 0.\, v_{5,4} + 2.27374\times 10^{-13}\, v_{5,5} - 1.7053\times 10^{-13}\, v_{5,6}\end{aligned}$$

```
In[19]:= % // Chop
Out[19]= 0
```

In this example, for $3 \le x \le 4$ and for $4 \le y \le 5$, the quantity `H[x,y]` can be expressed as the following polynomial of the independent variables x and y.

```
In[20]:= H3445[x, y];
```

If one does not suppress the output from the last line (which would produce a fairly long output), it becomes clear that all coefficients in the polynomial of the variables x and y are linear functions of the array $\texttt{VV} \in \mathbb{R}^{10\otimes 10}$. Consequently, if the variables x and y are integrated out, the expression will turn into a linear combination of all the entries (symbols) in the array `VV`.

Now we turn to concrete applications of the features described in this section. For the purpose of illustration, all time-consuming operations in this notebook are executed within the `Timing` function. The CPU times reflect the speed on Dual 2 GHz PowerPC processors with 2.5 GB of RAM. The *Mathematica* version is 6.0.1. For all inputs with computing time longer than 1 minute, a file that contains the returned value is made available.

■ Some Examples of Cubic Cubature Rules for Gaussian Integrals in $\mathbb{R}^2$

First, define the standard normal density in $\mathbb{R}^2$.

```
In[21]:= p[x_, y_] = 1/(2 π) e^(-(x^2+y^2)/2);
```

In this section, we develop cubature rules for computing integrals of the form

$$\int_{-\infty}^{\infty} d\,x \int_{-\infty}^{\infty} f(x,\, y)\, \boldsymbol{p}(x,\, y)\, d\,y$$

for certain functions $f(\cdot)$. We suppose that the last integral can be replaced by

$$\int_{-R}^{R} d\,x \int_{-R}^{R} f(x,\, y)\, \boldsymbol{p}(x,\, y)\, d\,y \tag{12}$$

without a significant loss of precision, where the value

```
In[22]:= R = 84/10;
```

is chosen so that

```
In[23]:= 1/Sqrt[2 π] Integrate[E^(-u^2/2), {u, R, ∞}] < $MachineEpsilon
Out[23]= True
```

The next step is to choose the abscissas $\mathtt{X}[\![k]\!]$, $1 \le k \le m$, and $\mathtt{Y}[\![l]\!]$, $1 \le l \le n$:

```
In[24]:= X = Table[x, {x, -R, R, 4/5}]; Y = X;
```

```
In[25]:= m = Length[X]; n = Length[Y];
```

For the sake of simplicity, in display equations we will write x_k instead of $\mathtt{X}[\![k]\!]$ and y_l instead of $\mathtt{Y}[\![l]\!]$. Then we define the associated array of symbolic values attached to the interpolation nodes (x_k, y_l)

```
In[26]:= V = Array[v_##1,##2 &, {m, n}];
```

and produce the actual `InterpolatingFunction` object:

```
In[27]:= f = ListInterpolation[Table[{X[[k]], Y[[l]], {V[[k, l]]}}, {k, 1, m}, {l, 1, n}]]
Out[27]= InterpolatingFunction[{{-42/5, 42/5}, {-42/5, 42/5}}, <>]
```

Note that `V` was defined simply as an array of symbols and that, therefore, `f` can be treated as a symbolic object, too. In any case, we have

$$\int_{-R}^{R} dx \int_{-R}^{R} f(x, y)\, p(x, y)\, dy = \sum_{k=1}^{m-1} \sum_{l=1}^{n-1} \int_{x_k}^{x_{k+1}} dx \int_{y_l}^{y_{l+1}} f(x, y)\, p(x, y)\, dy.$$

Note that each double integral on the right side of the last identity is actually an integral of some polynomial of the variables x and y multiplied by the standard normal density in $\mathbb{R}^2$. Furthermore, as was pointed out earlier, every coefficient in this polynomial is actually a linear function of the array `V`. Since the `InterpolatingFunction` object `f` was defined with `InterpolationOrder→3` (the default setting for `ListInterpolation`), we can write

$$\int_{x_k}^{x_{k+1}} dx \int_{y_l}^{y_{l+1}} f(x, y)\, p(x, y)\, dy = \sum_{i=0}^{3} \sum_{j=0}^{3} \ell_{k,l,i,j}(V) \int_{x_k}^{x_{k+1}} dx \int_{y_l}^{y_{l+1}} x^i\, y^j\, p(x, y)\, dy,$$

where $\ell_{k,l,i,j}(\cdot) : \mathbb{R}^{m\otimes n} \mapsto \mathbb{R}$ are linear functions of the tensor V that can be computed once and for all by using the method described in the previous section. In addition, the integrals in the right side of the last identity also can be computed once and for all by using straightforward integration—this is the only place in the procedure where actual integration takes place. Once these calculations are completed, the integral (12) can be expressed as an explicit linear function of the array `V` of the form

```
Flatten[Ω] · Flatten[V]
```

for some fixed tensor $\Omega \in \mathbb{R}^{m\otimes n}$. Of course, the actual calculation of the tensor Ω is much more involved and time consuming than the direct numerical evaluation of the integral (12) in those instances where `f` is not a symbolic object, that is, when the symbols $v_{k,l}$ are given concrete numeric values. The point is that the calculation of the tensor Ω is tantamount to writing the integral (12) as an explicit function of the symbols that define `f`; that is, we evaluate simultaneously the entire family of integrals for all possible choices of actual numeric values that can be assigned to the symbols $v_{k,l}$. Once the tensor Ω is calculated, we can write

$$\int_{-R}^{R} d\,x \int_{-R}^{R} f(x,\, y)\, p(x,\, y)\, d\,y = \texttt{Flatten[}\Omega\texttt{]} \cdot \texttt{Flatten[V]}. \tag{13}$$

Note that this identity is exact if the abscissas $(x_k,\, y_l)$ are exact numbers. In other words, by computing the tensor Ω we have made a *cubature rule*. Furthermore, when $F(\cdot)$ is some function defined in the rectangle $[-R,\, R]\times[-R,\, R]$ that has already been defined and the symbol `V` is given the numeric values

```
Table[F[X[[k]], Y[[l]]], {k, 1, m}, {l, 1, n}],
```

then the right side of equation (13) differs from the left side by no more than $4\,R^2$ times the uniform distance between the function $F(\cdot)$ and the interpolating function created from the assignment

$$(x_k,\, y_l) \longrightarrow F(x_k,\, y_l).$$

First, we compute—once and for all—all integrals of the form

$$\int_{x_k}^{x_{k+1}} d\,x \int_{y_l}^{y_{l+1}} x^i\, y^j\, \boldsymbol{p}(x,\, y)\, d\,y$$

for all choices of $1 \le k \le m$, $1 \le l \le n$, $0 \le i \le 3$, and $0 \le j \le 3$, and we represent these integrals as explicit functions of the symbols $\{x_k,\, x_{k+1},\, y_l,\, y_{l+1}\}$.

Instead of calculating the tensor `integrals`, one may load its (precomputed) value from the files integrals.mx or integrals.txt (if available):

```
<< "C:/LocalHDlocation/integrals.mx"
```

or

```
<< "C:/LocalHDlocation/integrals.txt"
```

```
In[28]:= Timing[
           integrals = Table[Integrate[x^i * y^j * p[x, y], {x, Xk, Xk1}, {y, Yl, Yl1}],
             {i, 0, 3}, {j, 0, 3}];]
Out[28]= {40.43, Null}
```

Next, we replace the symbols $\{x_k, x_{k+1}, y_l, y_{l+1}\}$ with the actual numeric values of the associated abscissas on the grid:

```
In[29]:= Timing[WM = Table[integrals /. {Xk → X[[k]], Xk1 → X[[k + 1]],
                    Yl → Y[[l]], Yl1 → Y[[l + 1]]}, {k, m - 1}, {l, n - 1}];]
Out[29]= {2.26533, Null}

In[30]:= Dimensions[WM]
Out[30]= {21, 21, 4, 4}
```

The next step is to create the lists of the midpoints between all neighboring abscissas

```
In[31]:= Xm = Table[1/2 * (X[[i]] + X[[i + 1]]), {i, 1, m - 1}];
         Ym = Table[1/2 * (Y[[i]] + Y[[i + 1]]), {i, 1, n - 1}];
```

which would allow us to produce the Taylor expansion of the `Interpolating`⋱`Function` object `f` at the points $(\texttt{Xm}[\![k]\!], \texttt{Ym}[\![l]\!]) \in \mathbb{R}^2$, for any $1 \le k \le m-1$ and any $1 \le l \le n-1$. As noted earlier, for every choice of k and l, the associated expansion coincides with the object `f` everywhere in the rectangle

$$[\texttt{X}[\![k]\!], \texttt{X}[\![k+1]\!]] \times [\texttt{Y}[\![l]\!], \texttt{Y}[\![l+1]\!]] \subset \mathbb{R}^2.$$

Now we will compute the entire expression

$$\sum_{k=1}^{m-1} \sum_{l=1}^{n-1} \left(\sum_{i,j=0}^{3} \ell_{k,l,i,j}(V) \int_{x_k}^{x_{k+1}} dx \int_{y_l}^{y_{l+1}} x^i y^j \, p(x, y) \, dy \right)$$

as a linear function of the symbols $v_{k,l}$, $1 \le k \le m$, $1 \le l \le n$. Note that for every fixed k and l, the third summation simply gives the dot product between the vectors

`Flatten[CoefficientList[Te[f, Xm[[k]], Ym[[l]]], {x, y}]]`
and `Flatten[WM[[k, l]]]`.

It is important to recognize that the sum of these dot products gives the *exact* value of the integral

$$\int_{-R}^{R} dx \int_{-R}^{R} f(x, y) \, p(x, y) \, dy.$$

Unfortunately, the exact values of the integrals stored in the list `WM` are quite cumbersome:

```
In[32]:= WM[[5, 7, 3, 1]]
```

$$\text{Out[32]}= \frac{1}{4\,e^{116/5}\sqrt{\pi}}\left(\operatorname{erf}\left(\frac{7\sqrt{2}}{5}\right)-\operatorname{erf}\left(\frac{9\sqrt{2}}{5}\right)\right)\left(\sqrt{2}\left(\frac{26\,e^{242/25}}{5}-\frac{22\,e^{338/25}}{5}\right)+e^{116/5}\sqrt{\pi}\left(\operatorname{erf}\left(\frac{11\sqrt{2}}{5}\right)-\operatorname{erf}\left(\frac{13\sqrt{2}}{5}\right)\right)\right)$$

Since working with such expressions becomes prohibitively slow, we convert them to floating-point numbers (we can, of course, use any precision that we wish).

```
In[33]:= Timing[WMN = N[WM, 20];]
```

```
Out[33]= {1.22097, Null}
```

Note that this last operation is the *only* source of errors in the integration of the interpolating function `f`.

Finally, we compute the integral

$$\int_{-R}^{R} d\,x \int_{-R}^{R} f(x,\,y)\,p(x,\,y)\,d\,y$$

in symbolic form, that is, as an *explicit* function of the symbols $v_{k,l}$ that define the interpolating function `f`—what we are going to compute are the actual numerical values of the coefficients for the symbols $v_{k,l}$ in the linear combination that represents this integral. Consequently, an expression in symbolic form for the integral is nothing but a tensor of actual numeric values. We again stress that the only loss of precision comes from the conversion of the exact numeric values for the integrals stored in `WM` into the floating-point numbers stored in `WMN`.

Instead of calculating the object `CoeffList` in the following, which contains the actual linear combination of the symbols $v_{i,j}$, one may load its (precomputed) value from the files CoeffList.mx or CoeffList.txt (if available):

```
<< "C:/LocalHDlocation/CoeffList.mx"
```

or

```
<< "C:/LocalHDlocation/CoeffList.txt"
```

```
In[34]:= Timing[CoeffList = Total[
            Table[Total[Expand[CoefficientList[Te(f, Xm[[i]], Ym[[j]]), {x, y}]
                  WMN[[i, j]]], 2], {i, m - 1}, {j, n - 1}], 2];]
```

```
Out[34]= {98.8673, Null}
```

Now we extract the numeric values for the coefficients $v_{k,l}$ and store those values in the tensor Ω:

```
In[35]:= Timing[Ω = Table[Coefficient[CoeffList, v_{k,l}], {k, m}, {l, n}];]
Out[35]= {0.119846, Null}
```

Note that in order to produce the actual cubature rule we need to compute the tensor Ω only once, so that several minutes of computing time is quite acceptable. Note also that Ω (i.e., the cubature rule) can be calculated with any precision. This means that if we approximate an integral of the form

$$\int_{-R}^{R} dx \int_{-R}^{R} u(x, y)\, p(x, y)\, dy$$

with the dot product

```
Flatten[Table[u〚X〚i〛, Y〚j〛〛, {i, m}, {j, n}]] · Flatten[Ω],
```

then the upper estimate for the error in this approximation can be made arbitrarily close to the uniform distance between the function $u(\cdot, \cdot)$ and the interpolating function created from the values that `u` takes on the grid multiplied by the area of the integration domain, which is $4R^2$.

Now we turn to some examples. First, consider the function

```
In[36]:= u[x_, y_] = x^2 + y^2;
```

and set

```
In[37]:= V = Table[u[X〚k〛, Y〚l〛], {k, m}, {l, n}];
```

Now we have

```
In[38]:= Flatten[V].Flatten[Ω]
Out[38]= 1.9999999999999934
```

```
In[39]:= NIntegrate[u[x, y] * 1/(2 π) * e^(-(x^2+y^2)/2), {x, -R, R}, {y, -R, R}]
Out[39]= 2.
```

```
In[40]:= Abs[% - %%]
Out[40]= 1.51149×10^-9
```

For the function

```
In[41]:= u[x_, y_] = ArcTan[x^2 * e^y]; V = Table[u[X〚k〛, Y〚l〛], {k, m}, {l, n}];
```

we get

```
In[42]:= Abs[Flatten[V].Flatten[Ω] -
            NIntegrate[u[x, y] * 1/(2 π) * e^(-(x^2+y^2)/2), {x, -R, R}, {y, -R, R}]]
Out[42]= 0.0086428
```

One must be aware that `NIntegrate` is a very efficient procedure and, generally, replacing `NIntegrate` with a "hand made" cubature rule of the type described in

this section may be justified only when the integrand cannot be defined in terms of standard functions, or when a better control of the error is needed, provided that one can control the uniform distance between the integrand and the respective interpolating function created from the grid. In general, such "hand made" cubatures become useful mostly in situations where one has to compute a very large number of integrals for similarly defined integrands—assuming, of course, that the uniform error from the interpolation in all integrands can be controlled (this would be the case, for example, if one has a global bound on a sufficient number of derivatives).

Interpolation quadrature rules for single integrals can be obtained in much the same way. It must be noted that, in general, spline quadrature rules are considerably more straightforward than spline cubature rules. This is because smooth interpolation in $\mathbb{R}^1$ can be obtained without any additional information about the derivatives at the grid points and this is not the case in spaces of dimension greater than or equal to 2. For example, cubic splines in $\mathbb{R}^1$ are uniquely determined by the values at the grid points and by the requirement that the second derivative vanishes at the first and last grid points (the so-called natural splines). In contrast, smooth interpolation in two or more dimensions depends on the choice of the gradient and at least one mixed derivative at every grid point, and when information about these quantities is not available—as is often the case—one usually resorts to divided difference interpolation. In general, the efficiency of the interpolation may be increased by using a nonuniform grid and by placing more grid points in the regions where the functions that are being interpolated are more "warped." We will illustrate this approach in the last section. The use of nonsmooth interpolation, such as bilinear interpolation in $\mathbb{R}^2$, for example, reduces considerably the computational complexity of the problem. In any case, the error from the interpolation must be controlled in one way or another. While the discussion of this issue is beyond the scope of this article, it should be noted that rather powerful estimates for the interpolation error are well known.

■ Some Examples of Bilinear Cubature Rules for Gaussian Integrals in $\mathbb{R}^2$

In general, just as one would expect, cubature rules based on bilinear interpolation are less accurate than the cubature rules described in the previous section (assuming, of course, one uses the same grid and keeping in mind that this claim is made just in general). However, from a computational point of view, bilinear cubature rules are considerably simpler and easier to implement, especially for more general integrals of the form

$$\int f(\phi(x))\, p(x)\, dx$$

for some fixed functions $x \longrightarrow \phi(x)$ and $x \longrightarrow p(x)$. The greater computational simplicity actually allows one to use a denser grid, which, in many cases, is a reasonable compensation for the loss in smoothness. The dynamic integration procedures in $\mathbb{R}^2$ that are discussed in the next section are all based on bilinear cubature rules (in the case of $\mathbb{R}^1$ we will use cubic quadratures).

Here we will be working with the same interpolation grid `X[[k]]`, $1 \le k \le m$, `Y[[l]]`, $1 \le l \le n$, that was defined in the previous section and, just as before, in all display formulas will write x_k instead of `X[[k]]` and y_l instead of `Y[[l]]`. Instead of working with the interpolation objects of the form

```
ListInterpolation[
  Table[{{X[[k]], Y[[l]]}, {v_{k,l}}}, {k, 1, m}, {l, 1, n}]],
```

we will work with bilinear interpolation objects that we are going to "construct from scratch" as explicit functions of the variables x and y that depend on the respective grid points and tabulated values. This means that for

$$x_k \le x \le x_{k+1} \qquad \text{and} \qquad y_l \le y \le y_{l+1},$$

the bilinear interpolation object can be expressed as

```
bi(x, y, X[[k]], X[[k + 1]], Y[[l]], Y[[l + 1]], v_{k,l}, v_{k,l+1}, v_{k+1,l}, v_{k+1,l+1}),
```

where the function `bi` is defined as

```
In[43]:= bi[x_, y_, x1_, x2_, y1_, y2_, V11_, V12_, V21_, V22_] =
  (1 - (x - x1)/(x2 - x1)) * (1 - (y - y1)/(y2 - y1)) * V11 + (x - x1)/(x2 - x1) * (1 - (y - y1)/(y2 - y1)) * V21 +
  (1 - (x - x1)/(x2 - x1)) * (y - y1)/(y2 - y1) * V12 + (x - x1)/(x2 - x1) * (y - y1)/(y2 - y1) * V22;
```

It is clear from this definition that, treated as a function of the variables x and y, the expression `bi` can be written as

$$A + B\,x + C\,y + D\,x\,y,$$

where the entire list $\{A, B, C, D\}$ can be treated as a function of the grid points $(x_k, x_{k+1}, y_l, y_{l+1})$ and the tabulated values $(v_{k,l}, v_{k,l+1}, v_{k+1,l}, v_{k+1,l+1})$. In fact, this function (that is, a list of functions) can be written as

```
In[44]:= cfbi[x1_, x2_, y1_, y2_, V11_, V12_, V21_, V22_] =
           CoefficientList[bi[x, y, x1, x2, y1, y2, V11, V12, V21, V22], {x, y}];
```

For every fixed k and l, the integral of the bilinear interpolating object taken over the rectangle

$$x_k \le x \le x_{k+1} \qquad \text{and} \qquad y_l \le y \le y_{l+1}$$

is simply the dot product between the list $\{A, B, C, D\}$ and the list of integrals (over the same rectangle) of the functions

$$(x, y) \longrightarrow p(x, y),\ (x, y) \longrightarrow x\,p(x, y),$$
$$(x, y) \longrightarrow y\,p(x, y),\ \text{and}\ (x, y) \longrightarrow x\,y\,p(x, y).$$

Now we will calculate—and store—all these lists (of integrals) of dimension 4 for all choices for $1 \le k \le m$ and for $1 \le l \le n$:

```
In[45]:= ip00[a_, b_, c_, d_] =
           Assuming[a ∈ ℝ && b ∈ ℝ && c ∈ ℝ && d ∈ ℝ && a < b && c < d,
             Integrate[p[x, y], {x, a, b}, {y, c, d}]]
```

$$\text{Out[45]=}\quad \frac{1}{4}\left(\operatorname{erf}\left(\frac{a}{\sqrt{2}}\right)-\operatorname{erf}\left(\frac{b}{\sqrt{2}}\right)\right)\left(\operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)-\operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right)$$

```
In[46]:= ip10[a_, b_, c_, d_] =
           Assuming[a ∈ ℝ && b ∈ ℝ && c ∈ ℝ && d ∈ ℝ && a < b && c < d,
             Integrate[x * p[x, y], {x, a, b}, {y, c, d}]]
```

$$\text{Out[46]=}\quad \frac{e^{-\frac{a^2}{4}-\frac{b^2}{4}}\left(\operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)-\operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right)\sinh\left(\frac{1}{4}\left(a^2-b^2\right)\right)}{\sqrt{2\pi}}$$

```
In[47]:= ip01[a_, b_, c_, d_] =
           Assuming[a ∈ ℝ && b ∈ ℝ && c ∈ ℝ && d ∈ ℝ && a < b && c < d,
             Integrate[y * p[x, y], {x, a, b}, {y, c, d}]]
```

$$\text{Out[47]=}\quad \frac{e^{-\frac{c^2}{4}-\frac{d^2}{4}}\left(\operatorname{erf}\left(\frac{a}{\sqrt{2}}\right)-\operatorname{erf}\left(\frac{b}{\sqrt{2}}\right)\right)\sinh\left(\frac{1}{4}\left(c^2-d^2\right)\right)}{\sqrt{2\pi}}$$

```
In[48]:= ip11[a_, b_, c_, d_] =
           Assuming[a ∈ ℝ && b ∈ ℝ && c ∈ ℝ && d ∈ ℝ && a < b && c < d,
             Integrate[x * y * p[x, y], {x, a, b}, {y, c, d}]]
```

$$\text{Out[48]=}\quad \frac{2\,e^{\frac{1}{4}\left(-a^2-b^2-c^2-d^2\right)}\sinh\left(\frac{1}{4}\left(a^2-b^2\right)\right)\sinh\left(\frac{1}{4}\left(c^2-d^2\right)\right)}{\pi}$$

```
In[49]:= Timing[
           WM2 = Table[{ip00[X[k], X[k + 1], Y[l], Y[l + 1]], ip01[X[k], X[
                   k + 1], Y[l], Y[l + 1]], ip10[X[k], X[k + 1], Y[l], Y[l + 1]],
               ip11[X[k], X[k + 1], Y[l], Y[l + 1]]}, {k, m - 1}, {l, n - 1}];]
Out[49]= {0.444583, Null}

In[50]:= Timing[WM2N = N[WM2, 20];]
Out[50]= {0.028574, Null}
```

and then compute the sum of all dot products:

```
In[51]:= Timing[
           loc = Total[Table[(Flatten[cfbi[X[k], X[k + 1], Y[l], Y[l + 1], v_{k,l}, v_{k,l+1},
                       v_{l+1,k}, v_{k+1,l+1}]].WM2N[k, l]) //
                   Simplify, {k, m - 1}, {l, n - 1}], 2];]
Out[51]= {0.488303, Null}
```

This sum is a linear function of the symbols $v_{k,l}$, and this linear function represents the integral of the interpolating function $f(\cdot)$ over the entire interpolation region (each dot product gives the integral of $f(\cdot)$ in the respective subregion). Finally, we must extract the coefficients for the symbols $v_{k,l}$ from the linear combination (of those symbols) that we just obtained.

```
In[52]:= Timing[Θ = Table[∂_{v_{k,l}} (loc), {k, m}, {l, n}];]
Out[52]= {0.643791, Null}
```

Now we can use the tensor Θ in the same way in which we used the tensor Ω in the previous section: although Θ contains concrete numeric values, this tensor still has the meaning of "integral of $f(\cdot)$ in symbolic form." For example, if we set

```
In[53]:= u[x_, y_] = e^(0.1 x y);
```

then we obtain

```
In[54]:= Timing[V = Table[u[X[[k]], Y[[l]]], {k, m}, {l, n}]; Flatten[V].Flatten[Θ]]
Out[54]= {0.001992, 1.0014}
```

which is reasonably close to

```
In[55]:= NIntegrate[u[x, y] * 1/(2 π) * e^(-(x^2+y^2)/2), {x, -R, R}, {y, -R, R}] // Timing
Out[55]= {0.045307, 1.00504}
```

In this case, the cubic cubature rule is considerably more accurate:

```
In[56]:= Timing[V = Table[u[X[[k]], Y[[l]]], {k, m}, {l, m}]; Flatten[V].Flatten[Ω]]
Out[56]= {0.00184, 1.00503}
```

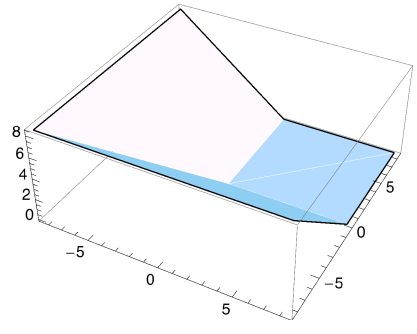
Here is another example.

```
In[57]:= u[x_, y_] = Max[R - Min[x + R, y + R], 0];
```

This function is nonsmooth, as the following 3D plot shows.

```
In[58]:= Plot3D[u[x, y], {x, -R, R}, {y, -R, R},
           PlotPoints → 150, Mesh → False, PlotRange → All]
```

Out[58]=



```
In[59]:= Timing[V = Table[u[X[[k]], Y[[l]]], {k, m}, {l, n}]; Flatten[V].Flatten[Θ]]
Out[59]= {0.003845, 0.744238893601963570}

In[60]:= Timing[V = Table[u[X[[k]], Y[[l]]], {k, m}, {l, n}]; Flatten[V].Flatten[Ω]]
Out[60]= {0.003853, 0.6874360084736024}

In[61]:= NIntegrate[u[x, y] * 1/(2 π) * e^(-(x^2+y^2)/2), {x, -R, R}, {y, -R, R}] // Timing
Out[61]= {0.151764, 0.681037}

In[62]:= Abs[{%[[2]] - %%[[2]], %[[2]] - %%%[[2]]}]
Out[62]= {0.00639893, 0.0632018}
```

Finally, we remark that—at least in principle—the procedures developed in this and in the previous sections may be used with any other, that is, nonGaussian, probability density $\boldsymbol{p}(\cdot)$, including nonsmooth densities.

■ Application to Optimal Stopping Problems and Option Pricing

The main objective in this section is to revisit the method of dynamic interpolation and integration described in [1] and to present a considerably more efficient implementation of this method, which takes advantage of the fact that the objects that are being integrated sequentially are actually interpolating functions. More importantly, we will show how to implement this method in the case of optimal stopping problems for certain diffusions in $\mathbb{R}^2$. Essentially, this section is about the actual implementation in *Mathematica* of the method encoded in the *space-discretized version* of the *time-discrete dynamic programming equation* (7). The solution is approximated in terms of dynamic cubature rules of the form (8) by using the procedure encoded in equation (11). For the sake of simplicity, the examples presented in this section will be confined to the case where the (diffusion type) observation process $(X_t)_{t \geq 0}$ is computable, in the sense that for any fixed $T \geq t \geq 0$, the random variable X_T can be expressed as

$$X_T = \psi_{T-t}(X_t, W_T - W_t)$$

for some computable function $\psi_{T-t} : \mathbb{R}^N \times \mathbb{R}^N \mapsto \mathbb{R}^N$ (note that the position of the diffusion process at time t, X_t, is independent of the future increment of the Brownian motion $W_T - W_t$). In other words, the conditional law of X_T given the event $\{X_T = x\}$ can be identified with the distribution law of a random variable of the form

$$\psi_{T-t}\left(x, \sqrt{T-t}\ \Gamma\right),$$

where Γ is a standard normal $\mathbb{R}^N$-valued random variable. Of course, when $(X_t)_{t \geq 0}$ is computable in the sense that we just described, then there is no need to

use the approximation (6), and, if $(X_t)_{t \geq 0}$ is not computable, in principle at least, the procedures described in this section can still be used, provided that the definition of the function $\psi_{T-t}(\cdot, \cdot)$ is changed according to the recipe (6).

We note that the *Mathematica* code included in this section is independent of the code included in all previous sections.

To begin with, define the following symmetric probability density function in $\mathbb{R}^2$.

```
In[63]:= p[x_, y_] = 1/(2 π) * e^-((x^2 + y^2)/2);
```

This is nothing but the probability density of a bivariate normal random variable $(X, Y) \in \mathbb{R}^2$, with standard normal marginals X and Y and with correlation $\mathrm{E}[X\,Y] = 0$. Let W_t, $t \geq 0$, and B_t, $t \geq 0$, be two standard Brownian motions that are independent, so that $\mathrm{E}[W_t\, B_t] = 0$ for any $t \geq 0$. Next, for some fixed $\sigma > 0$ and for $(x, y) \in \mathbb{R}^2$, set $a_t^x = x\, e^{\sigma W_t}$ and $b_t^y = y\, e^{\sigma B_t}$ for any $t \geq 0$. Note that the joint distribution law of $\left(a_t^x, b_t^y\right)$ is the same as the joint distribution law of $\left(x\, e^{\sigma \sqrt{t}\, X},\ y\, e^{\sigma \sqrt{t}\, Y}\right)$. It is a trivial matter to check that

$$\mathbb{R}_+ \ni t \longrightarrow \left(a_t^x, b_t^y\right) \in \mathbb{R}^2$$

is a diffusion process in $\mathbb{R}^2$ with generator

$$\mathcal{L} = \frac{\sigma^2}{2}\left(x^2\, \partial_{x,x} + y^2\, \partial_{y,y}\right).$$

In what follows, we study the problem for optimal stopping of the process $\left(a_t^x, b_t^y\right)_{t \geq 0}$ no later than time $T > 0$, for a given termination payoff $\Lambda : \mathbb{R}_+^2 \mapsto \mathbb{R}$. For the sake of simplicity, we will suppose that the discount rate for future payoffs is 0. In the special case where $\Lambda(x, y) = \lambda(x)$, $x \geq 0$, for some "sufficiently nice" function of one variable $\lambda : \mathbb{R}_+ \mapsto \mathbb{R}$, the problem comes down to the optimal stopping of the one-dimensional diffusion $\left(a_t^x\right)_{t \geq 0}$. We analyze this one-dimensional case first.

```
In[64]:= N[ξ_] = 1/Sqrt[2 π] e^-(ξ^2/2);
```

```
In[65]:= σ = 0.3;
```

The time step in the procedure, that is, the quantity $\frac{T}{v}$, is set to

```
In[66]:= t = .05;
```

Next, define the interpolation grid:

```
In[67]:= X = Join[Table[x // N, {x, 2, 32, 2}], Table[x // N, {x, 34, 42, 2}],
            Table[x // N, {x, 44, 80, 4}], {90, 100}, Table[x // N, {x, 120, 200, 20}]];
          m = Length[X]; XX = Join[{$MachineEpsilon}, X, {300, 400}];
          Xm = Table[1/2 * (XX[[k]] + XX[[k + 1]]), {k, 1, m + 2}];
```

Just as before, in all display equations we will write x_k instead of `X[[k]]` and $\overline{x}_k$ instead of `XX[[k]]`. The boundary conditions will be prescribed at the first and last two abscissas on the extended grid `XX`. This means that the recursive procedure will update only the values on the grid `X`. Thus, the procedure that we develop can be applied only to situations where the value function does not change at those points in the state-space that are either close to 0 or happen to be very large.

The next step is to define the list of tabulated symbols

```
In[68]:= vrsb = Array[b_#1 &, {m + 3}];
```

and the associated (symbolic) `InterpolatingFunction` object

```
In[69]:= g = ListInterpolation[Table[{XX[[k]], {vrsb[[k]]}}, {k, 1, m + 3}]];
```

Next, extract the coefficients of the cubic polynomials that represent the object `g` between the interpolation nodes.

```
In[70]:= te[ff_, x0_] := Sum[1/(i!) * ff^(i)(x0) * (x - x0)^i, {i, 0, 3}];
```

```
In[71]:= m0loc = Table[CoefficientList[te[g, Xm[[k]]], {x}] // Simplify, {k, m + 2}];
```

For example,

```
In[72]:= m0loc[[1]] // Chop
```

```
Out[72]= {1. b1, -0.916667 b1 + 1.5 b2 - 0.75 b3 + 0.166667 b4,
           0.25 b1 - 0.625 b2 + 0.5 b3 - 0.125 b4,
           -0.0208333 b1 + 0.0625 b2 - 0.0625 b3 + 0.0208333 b4}
```

which means that for $\overline{x}_1 \le x \le \overline{x}_2$, the expression `g[x]` can be identified with the polynomial

```
In[73]:= {1, x, x^2, x^3}.m0loc[[1]]
```

```
Out[73]= (-0.0208333 b1 + 0.0625 b2 - 0.0625 b3 + 0.0208333 b4) x^3 +
          (0.25 b1 - 0.625 b2 + 0.5 b3 - 0.125 b4) x^2 +
          (-0.916667 b1 + 1.5 b2 - 0.75 b3 + 0.166667 b4) x + 1. b1 -
          4.44089 10^-16 b2 + 1.11022 10^-16 b3 - 1.38778 10^-17 b4
```

Next, note that the relation

$$\overline{x}_k \le x_\xi \, e^{\sigma \sqrt{t}\, x} \le \overline{x}_{k+1}$$

is equivalent to

$$L(\xi,\, k) \equiv \frac{1}{\sigma\sqrt{t}} \log\left(\frac{\overline{x}_k}{x_\xi}\right) \le x \le \frac{1}{\sigma\sqrt{t}} \log\left(\frac{\overline{x}_{k+1}}{x_\xi}\right) \equiv L(\xi,\, k+1).$$

For every fixed $\xi = 1, \ldots, m$, we will compute all integrals

$$\int_{L(\xi,k)}^{L(\xi,k+1)} e^{(i-1)\,\sigma\sqrt{t}\;x}\, \mathcal{N}(x)\, dx, \qquad 1 \le i \le 4, \qquad 1 \le k \le m+2,$$

and will store these integrals in the matrix (initially filled with zeros):

```
In[74]:= wl = Table[(Table[0, {i, 0, 3}]), {k, 1, m + 2}];
```

In particular, the integral

$$\int_{L(\xi,k)}^{L(\xi,k+1)} g\left(x_\xi\, e^{\sigma\sqrt{t}\;x}\right) \mathcal{N}(x)\, dx$$

can be replaced by

$$\left\{1,\, x_\xi,\, x_\xi^2,\, x_\xi^3\right\} \cdot \left(\texttt{m0loc}\left[\!\left[k\right]\!\right] * \texttt{wl}\left[\!\left[k\right]\!\right]\right),$$

and we remark that

$$\texttt{m0loc}\left[\!\left[k\right]\!\right] * \texttt{wl}\left[\!\left[k\right]\!\right] \equiv \sum_{i=1}^{4} \texttt{m0loc}\left[\!\left[k,\, i\right]\!\right] * \texttt{wl}\left[\!\left[k,\, i\right]\!\right].$$

The entire expression

$$\int_{L(\xi,1)}^{L(\xi,m+3)} g\left(x_\xi\, e^{\sigma\sqrt{t}\;x}\right) \mathcal{N}(x)\, dx = \sum_{k=1}^{m+2} \left\{1,\, x_\xi,\, x_\xi^2,\, x_\xi^3\right\}.\left(\texttt{m0loc}\left[\!\left[k\right]\!\right] * \texttt{wl}\left[\!\left[k\right]\!\right]\right)$$

then becomes a linear function of the symbols $\{b_1, \ldots, b_{m+3}\}$. This linear function (i.e., vector of dimension $m + 3$) depends on the index ξ and represents a quadrature rule for the integral in the left side in the last identity. We stress that we need one such rule for every abscissa x_ξ, $1 \le \xi \le m$. We will store all these rules (i.e., vectors of dimension $m + 3$) in the tensor (initially filled with zeroes):

```
In[75]:= ω = Table[Table[0, {k, 1, m + 3}], {ξ, 1, m}];

In[76]:= Clear[ll, ul]; locInt =
           Table[Integrate[e^((i-1)*σ*Sqrt[t]*x) * N[x], {x, ll, ul}] // Simplify, {i, 1, 4}];
```

```
In[77]:= Do[(Do[Do[(wl[[k, i]] = N[(locInt[[i]] /. {ll → 1/(σ * Sqrt[t]) * Log[XX[[k]]/X[[ξ]]],
                                                    ul → 1/(σ * Sqrt[t]) * Log[XX[[k + 1]]/X[[ξ]]]})]),
                                {i, 1, 4}], {k, m + 2}];
          lcls = Sum[({1, X[[ξ]], X[[ξ]]^2, X[[ξ]]^3}.(m0loc[[k]] * wl[[k]]) // Simplify),
                     {k, m + 2}];
          ω[[ξ]] = Table[∂_(b_k) (lcls), {k, m + 3}];), {ξ, m}] // Timing
```

```
Out[77]= {0.90158, Null}
```

As a first application, we compute the price of an American put option with payoff:

```
In[78]:= Λ[x_] = Max[40 − x, 0]
```

```
Out[78]= max(0, 40 − x)
```

The first iteration will be done by using a direct numerical integration—not interpolation quadrature rules.

```
In[79]:= Timing[
           V1 = Table[{Max[Λ[XX[[k]]], NIntegrate[Λ[XX[[k]] * E^(σ*Sqrt[t] *x)] * N(x),
                      {x, −∞, 1/(σ * Sqrt[t]) Log[40/XX[[k]]]}, MaxRecursion → 10,
                      AccuracyGoal → 16, SingularityDepth → 10]]}, {k, 1, m + 3}];]
```

```
Out[79]= {1.02531, Null}
```

Now we will do 19 additional iterations by using interpolation quadratures. The time step is $t = 0.05$ years. This will give us the approximate price of an American put with strike price 40 and one year left to expiry. Note that the list $\omega[\![k]\!]$ gives the integral associated with the abscissa x_k, which is the same as $\overline{x}_{k+1}$ for $1 \le k \le m$. The idea is to keep the value at the abscissa $\overline{x}_1$ fixed, throughout the iterations, equal to the initial value 40 and, similarly, keep the values at the abscissas $\overline{x}_{m+2}$ and $\overline{x}_{m+3}$ forever fixed at the initial values 0 and 0.

```
In[80]:= K = V1; VL = K;
```

```
In[81]:= Timing[
           Do[(Do[VL[[k + 1]] = {Max[Λ[ XX[[k + 1]], Flatten[K].ω[[k]]]}, {k, m}];
               K = VL), {19}];]
```

```
Out[81]= {0.010863, Null}
```

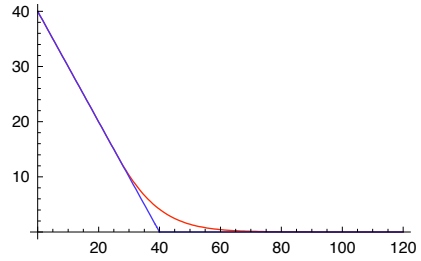
Note that the first iteration took about 20 times longer to compute than the remaining 19. Now we define the value function after 20 iterations—this is nothing but the option price with one year left to maturity treated as a function defined on the entire range of observable stock prices.

```
In[82]:= f = ListInterpolation[Table[{ XX[[k]], K[[k]]}, {k, 1, m + 3}]];
```

We plot the value function $f(\cdot)$ together with the termination payoff $\Lambda(\cdot)$.

```
In[83]:= Plot[{f[x], Λ[x]}, {x, 0, 120}, PlotRange → All,
           PlotPoints → 150, PlotStyle → {Hue[0.], Hue[0.7]}]
```

Out[83]=



The range of abscissas at which the value function and the termination payoff coincide is precisely the price range where immediate exercise of the option is optimal, assuming that there is exactly one year left to expiry.

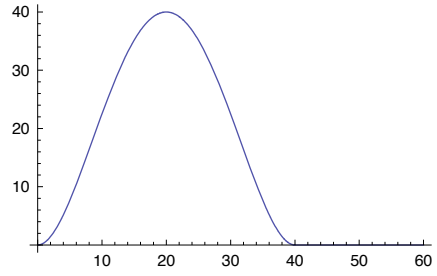
Now we consider the same optimal stopping problem but with a smooth payoff function, namely

```
In[84]:= Λ[x_] = 1/100 x^2 * 1/40 (Max[40 - x, 0])^2;
```

Essentially, this is an example of some sort of a generalized obstacle problem.

```
In[85]:= Plot[Λ[x], {x, 0, 60}, PlotRange → All]
```

Out[85]=



In this case, there is no need to perform the first iteration by using direct numerical integration.

```
In[86]:= Timing[V1 = Table[{Λ[XX[[k]]}, {k, 1, m + 3}];]
```

```
Out[86]= {0.000257, Null}
```

Now we perform 60 iterations, which will give us the value function with three years left to the termination deadline.

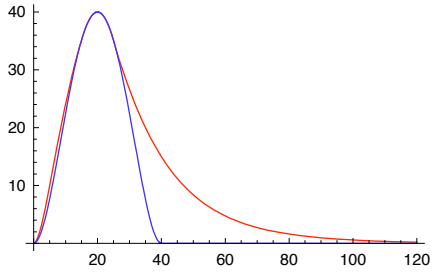
```
In[87]:= K = V1; VL = K;
```

```
In[88]:= Timing[
           Do[(Do[VL[[k + 1]] = {Max[Λ[XX[[k + 1]]], Flatten[K].ω[[k]]}, {k, 1, m}];
               K = VL), {60}];]
```

```
Out[88]= {0.039661, Null}
```

```
In[89]:= f = ListInterpolation[Table[{XX[[k]], K[[k]]}, {k, 1, m + 3}]];

In[90]:= Plot[{f[x], Λ[x]}, {x, 0, 120},
           PlotRange → All, PlotStyle → {Hue[0.], Hue[0.7]}]
```

Out[90]=



The most common approach to dynamic programing problems of this type is to use the free boundary problem formulation, followed by some suitable variation of the finite difference scheme. However, such an approach is much less straightforward and usually takes much longer to compute—a fraction of a second would be very hard to imagine even on a very powerful computer. In fact, we could double the number of the interpolation nodes and cut the time step in half, which would roughly quadruple the timing, and still complete the calculation within one second.

It is important to point out that, in terms of the previous procedure, there is nothing special about the standard normal density. In fact, we could have used just about any other reasonably behaved probability density function $\mathcal{N}(\cdot)$—as long as the associated integrals are computable. In particular, one can use this procedure in the context of financial models in which the pricing process is driven by some nonwhite noise process, including a noise process with jumps. The only requirement is that we must be able to compute the integrals

$$\int_{L(\xi,k)}^{L(\xi,k+1)} e^{(i-1)\,\sigma\sqrt{t}\,x}\,\mathcal{N}(x)\,dx, \qquad 1 \le i \le 4, \qquad 1 \le k \le m+2,$$

for any $\xi = 1, \ldots, m$.

Finally, we consider the two-dimensional case. We use the same grid-points on both axes:

```
In[91]:= Y = X; n = Length[Y]; YY = XX;
```

Although the lists `Y` and `YY` are the same as `X` and `XX`, in the notation that we use we will pretend that these lists might be different and will write y_k instead of `Y[[k]]`, $\overline{y}_k$ instead of `YY[[k]]`, and set

$$M(\eta, l) \equiv \frac{1}{\sigma\sqrt{t}} \log\left(\frac{\overline{y}_l}{y_\eta}\right).$$

We will use bilinear cubature rules, which are easier to implement—we need one such rule for every point (x_ξ, y_η), $1 \le \xi \le m$, $1 \le \eta \le n$. Thus, we must compute all integrals

$$\int_{L(\xi,k)}^{L(\xi,k+1)} dx \int_{M(\eta,l)}^{M(\eta,l+1)} e^{(i-1)\sigma\sqrt{t}\,x+(j-1)\sigma\sqrt{t}\,y}\, p(x, y)\, dy \tag{14}$$

for $i, j = 1, 2$, for $1 \le k \le m+2$, for $1 \le l \le n+2$, for $1 \le \xi \le m$, and for $\xi \le \eta \le n$ (because of the symmetry in the density $p(\cdot)$, it is enough to consider only the case $\eta \ge \xi$). Now we need to operate with substantially larger lists of data and for that reason we need to organize the *Mathematica* code differently.

Assuming that $(x, y) \longrightarrow f(x, y)$ is an interpolating function constructed by way of bilinear interpolation from the tabulated symbols

$$a_{k,l} \equiv f(\overline{x}_k, \overline{y}_l), \qquad 1 \le k \le m+3, \qquad 1 \le l \le n+3,$$

we can express each of the integrals

$$\int_{L(\xi,k)}^{L(\xi,k+1)} dx \int_{M(\eta,l)}^{M(\eta,l+1)} f\left(x_\xi\, e^{\sigma\sqrt{t}\,x},\ y_\eta\, e^{\sigma\sqrt{t}\,y}\right) p(x, y)\, dy$$

in the form

$$\{1, x_\xi\}.A_{\xi,\eta,k,l}.\{1, y_\eta\},$$

where $A_{\xi,\eta,k,l}$ is the 2×2 matrix given by (here we use the definition of `cfbi` from the previous section)

$$A_{\xi,\eta,k,l} = \texttt{cfbi}\left[\overline{x}_k,\ \overline{x}_{k+1},\ \overline{y}_l,\ \overline{y}_{l+1}, a_{k,l}, a_{k,l+1}, a_{k+1,l}, a_{k+1,l+1}\right] * \texttt{locIntegrals},$$

and `locIntegrals` is the 2×2 matrix of the integrals (14) indexed by $1 \le i, j \le 2$ (i is the first index and j is the second). In fact, with our special choice for the bivariate density $p(x, y)$, all these integrals can be calculated in closed form:

```
In[92]:= Clear[σ, t, x, y, lx, Lx, ly, Ly]

In[93]:= i = 1; j = 1; Integrate[e^((i-1) σ Sqrt[t] x + (j-1) σ Sqrt[t] y) * p[x, y], {x, lx, Lx}, {y, ly, Ly}]

Out[93]= 1/4 (Erf[lx/Sqrt[2]] - Erf[Lx/Sqrt[2]]) (Erf[ly/Sqrt[2]] - Erf[Ly/Sqrt[2]])

In[94]:= i = 2; j = 1; Integrate[e^((i-1) σ Sqrt[t] x + (j-1) σ Sqrt[t] y) * p[x, y], {x, lx, Lx}, {y, ly, Ly}]

Out[94]= 1/4 e^(t σ^2/2) (Erf[ly/Sqrt[2]] - Erf[Ly/Sqrt[2]]) (Erf[(lx - Sqrt[t] σ)/Sqrt[2]] - Erf[(Lx - Sqrt[t] σ)/Sqrt[2]])
```

```
In[95]:= i = 1; j = 2; Integrate[e^((i-1) σ √t x + (j-1) σ √t y) * p(x, y), {x, lx, Lx}, {y, ly, Ly}]
```

$$\text{Out[95]}= \frac{1}{4} e^{\frac{t\sigma^2}{2}} \left(\text{erf}\left(\frac{\text{lx}}{\sqrt{2}}\right) - \text{erf}\left(\frac{\text{Lx}}{\sqrt{2}}\right)\right)\left(\text{erf}\left(\frac{\text{ly} - \sqrt{t}\,\sigma}{\sqrt{2}}\right) - \text{erf}\left(\frac{\text{Ly} - \sqrt{t}\,\sigma}{\sqrt{2}}\right)\right)$$

```
In[96]:= i = 2; j = 2; Integrate[e^((i-1) σ √t x + (j-1) σ √t y) * p(x, y), {x, lx, Lx}, {y, ly, Ly}]
```

$$\text{Out[96]}= \frac{1}{4} e^{t\sigma^2} \left(\text{erf}\left(\frac{\text{lx} - \sqrt{t}\,\sigma}{\sqrt{2}}\right) - \text{erf}\left(\frac{\text{Lx} - \sqrt{t}\,\sigma}{\sqrt{2}}\right)\right)\left(\text{erf}\left(\frac{\text{ly} - \sqrt{t}\,\sigma}{\sqrt{2}}\right) - \text{erf}\left(\frac{\text{Ly} - \sqrt{t}\,\sigma}{\sqrt{2}}\right)\right)$$

After fixing the values for the volatility σ and the time step t

```
In[97]:= Clear[i, j]; σ = 0.3; t = .05;
```

the integrals can be written as explicit functions of the integration limits:

```
In[98]:= cip11 = Compile[{{lx, _Real}, {Lx, _Real}, {ly, _Real}, {Ly, _Real}},
           1/4 * (erf(lx/√2) - erf(Lx/√2)) * (erf(ly/√2) - erf(Ly/√2))];
```

```
In[99]:= cip21 =
           Compile[{{lx, _Real}, {Lx, _Real}, {ly, _Real}, {Ly, _Real}}, 1/4 * e^(t σ^2/2) *
             (erf(ly/√2) - erf(Ly/√2)) * (erf((lx - √t σ)/√2) - erf((Lx - √t σ)/√2))];
```

```
In[100]:= cip12 =
            Compile[{{lx, _Real}, {Lx, _Real}, {ly, _Real}, {Ly, _Real}}, 1/4 * e^(t σ^2/2) *
              (erf(lx/√2) - erf(Lx/√2)) * (erf((ly - √t σ)/√2) - erf((Ly - √t σ)/√2))];
```

```
In[101]:= cip22 = Compile[{{lx, _Real}, {Lx, _Real}, {ly, _Real}, {Ly, _Real}},
            1/4 * e^(t σ^2) * (erf((lx - √t σ)/√2) - erf((Lx - √t σ)/√2)) *
              (erf((ly - √t σ)/√2) - erf((Ly - √t σ)/√2))];
```

Our method depends in a crucial way on the fact that $\{1, X[\![\xi]\!]\}.A_{\xi,\eta,k,l}.\{1, Y[\![\eta]\!]\}$ is actually a linear function of the symbols $\{a_{k,l}, a_{k,l+1}, a_{k+1,l}, a_{k+1,l+1}\}$. In fact, we can compute the coefficients in this linear combination explicitly, as we now demonstrate.

```
In[102]:= Clear[a, b, c, d]

In[103]:= locIntegrals = {{C11, C12}, {C21, C22}}
```

$$\text{Out[103]=} \begin{pmatrix} C11 & C12 \\ C21 & C22 \end{pmatrix}$$

The next two definitions are taken from the previous section, so that the code included in this section can be self-contained.

```
In[104]:= bi[x_, y_, x1_, x2_, y1_, y2_, V11_, V12_, V21_, V22_] =
            (1 - (x - x1)/(x2 - x1)) * (1 - (y - y1)/(y2 - y1)) * V11 + (x - x1)/(x2 - x1) * (1 - (y - y1)/(y2 - y1)) * V21 +
              (1 - (x - x1)/(x2 - x1)) * (y - y1)/(y2 - y1) * V12 + (x - x1)/(x2 - x1) * (y - y1)/(y2 - y1) * V22;

In[105]:= cfbi[x1_, x2_, y1_, y2_, V11_, V12_, V21_, V22_] =
            CoefficientList[bi[x, y, x1, x2, y1, y2, V11, V12, V21, V22], {x, y}];

In[106]:= Coefficient[{1, xx}.(cfbi[x1, x2, y1, y2, a, b, c, d] * locIntegrals).{1, yy}, a] // Simplify
```

$$\text{Out[106]=} \frac{C11\,x2\,y2 - C21\,xx\,y2 - C12\,x2\,yy + C22\,xx\,yy}{(x1 - x2)(y1 - y2)}$$

```
In[107]:= Coefficient[{1, xx}.(cfbi[x1, x2, y1, y2, a, b, c, d] * locIntegrals).{1, yy}, b] // Simplify
```

$$\text{Out[107]=} \frac{-C11\,x2\,y1 + C21\,xx\,y1 + C12\,x2\,yy - C22\,xx\,yy}{(x1 - x2)(y1 - y2)}$$

```
In[108]:= Coefficient[{1, xx}.(cfbi[x1, x2, y1, y2, a, b, c, d] * locIntegrals).{1, yy}, c] // Simplify
```

$$\text{Out[108]=} \frac{-C11\,x1\,y2 + C21\,xx\,y2 + C12\,x1\,yy - C22\,xx\,yy}{(x1 - x2)(y1 - y2)}$$

```
In[109]:= Coefficient[{1, xx}.(cfbi[x1, x2, y1, y2, a, b, c, d] * locIntegrals).{1, yy}, d] // Simplify
```

$$\text{Out[109]=} \frac{C11\,x1\,y1 - C21\,xx\,y1 - C12\,x1\,yy + C22\,xx\,yy}{(x1 - x2)(y1 - y2)}$$

Thus, the entire integral

$$\int_{L(\xi,1)}^{L(\xi,m+3)} dx \int_{M(\eta,1)}^{M(\eta,n+3)} f\left(X[\![\xi]\!]\, e^{\sigma\sqrt{t}\,x},\, Y[\![\eta]\!]\, e^{\sigma\sqrt{t}\,y}\right) p(x,\, y)\, dy \tag{15}$$

can be treated as a linear function of the symbols

$$\left\{a_{k,l};\ 1 \le k \le m+3,\ 1 \le l \le n+3\right\}.$$

This linear function is nothing but a tensor of dimension $(m+3)\times(n+3)$ and this tensor is nothing but a cubature rule for the integral (15), which, obviously, depends on the indices ξ and η. Consequently, we have one such cubature rule (tensor) for every choice of $\xi = 1, \ldots, m$ and every choice of $\eta = \xi, \ldots, n$, and all such rules will be stored in the tensor (initially filled with zeros):

```
In[110]:= Ξ = Table[Table[0, {k, m + 3}, {l, n + 3}], {ξ, m}, {η, ξ, n}];
```

The calculation of the tensor Ξ is the key step in the implementation of the method of dynamic interpolation and integration in dimension 2. This is the place where we must face the "curse of the dimension." It is no longer efficient to express the global integral (15) as the sum of local integrals of the form (14) and then extract from the sum the coefficients for the symbols $a_{k,l}$. It turns out to be much faster if one updates the coefficients for the symbols $a_{k,l}$ sequentially, as the local integrals in (14) are calculated one by one. Just as one would expect, dealing with the boundary conditions in higher dimensions is also trickier. Throughout the dynamic integration procedure we will be updating the values of the value function at the grid points $\left(x_\xi,\, y_\eta\right)$, $1 \le \xi \le m$, $1 \le \eta \le n$, or, which amounts to the same, the grid points $\left(\overline{x}_k,\, \overline{y}_l\right)$, $2 \le k \le m+1$, $2 \le \eta \le n+1$. The values on the remaining grid points $\left(\overline{x}_k,\, \overline{y}_l\right)$ with $k = 1$, or $k = m+2$, or $k = m+3$, or $l = 1$, or $l = n+2$, or $l = n+3$, will be kept unchanged, or, depending on the nature of the problem, will be updated according to some other—one-dimensional, perhaps—dynamic quadrature rule.

Now we turn to the actual computation of the tensor Ξ. On a single "generic" processor, this task takes about $\frac{1}{2}$ hour (if more processors are available the task can be distributed between several different *Mathematica* kernels in a trivial way). *The key point is that this is a calculation that we do once and for all.* As will be illustrated shortly, once the tensor Ξ is calculated, a whole slew of optimal stopping problems can be solved within seconds. Of course, the "slew of optimal stopping problems" is limited to the ones where the termination payoff and the value functions obtained throughout the dynamic integration can be approximated reasonably well by way of bilinear interpolation from the same interpolation grid. In general, the set of these "quickly solvable" problems can be increased by choosing a denser grid and/or a grid that covers a larger area. However, doing so may become quite expensive. For example, if we were to double the number of grid points in each coordinate, the computing time for the tensor Ξ would increase roughly 16 times—again, everything is easily parallelizable.

It is important to recognize that the calculation of the list Ξ involves only the evaluation of standard functions and accessing the elements of a fairly large tensor (Ξ has a total of 1,245,621 entries). This is close to the "minimal number of calculations" that one may expect to get away with: if the number of discrete data from which the value function can be restored with a reasonable degree of accuracy is fairly large, then there is a fairly large number of values that will have to be calculated no matter what. Furthermore, the evaluation of standard functions is very close to the fastest numerical procedure that one may hope to get away with in this situation. From an algorithmic point of view, the efficiency of this procedure can be improved in two ways: (1) find a new way to encrypt the value function with fewer discrete values; and/or (2) represent all functions involved in the procedure (i.e., all combinations of standard functions) in a form that allows for an even faster evaluation. In terms of hardware, the speed of the procedure depends not only on the speed of the processor but also on the speed and the organization of the memory.

Instead of calculating the list Ξ, one may load its (precomputed) value from the files Xi.mx or Xi.txt (if available):

```
<< "C:/LocalHDlocation/Xi.mx"
```

or

```
<< "C:/LocalHDlocation/Xi.txt"
```

In[111]:=

```
Timing[Do[xx = X[[ξ]]; yy = Y[[η]];
 (Do[(lx = 1/(σ * Sqrt[t]) * Log[XX[[k]]/X[[ξ]]]; Lx = 1/(σ * Sqrt[t]) * Log[XX[[k + 1]]/X[[ξ]]];
   x1 = XX[[k]]; x2 = XX[[k + 1]];
   Do[(ly = 1/(σ * Sqrt[t]) * Log[YY[[l]]/Y[[η]]]; Ly =
      1/(σ * Sqrt[t]) * Log[YY[[l + 1]]/Y[[η]]]; y1 = YY[[l]]; y2 = YY[[l + 1]];
    C11 = cip11[lx, Lx, ly, Ly]; C12 = cip12[lx, Lx, ly, Ly];
    C21 = cip21[lx, Lx, ly, Ly]; C22 = cip22[lx, Lx, ly, Ly];
    Ξ[[ξ, η - ξ + 1, k, l]] = Ξ[[ξ, η - ξ + 1, k, l]] +
      (C11 x2 y2 - C21 xx y2 - C12 x2 yy + C22 xx yy)/((x1 - x2) (y1 - y2));
    Ξ[[ξ, η - ξ + 1, k, l + 1]] = Ξ[[ξ, η - ξ + 1, k, l + 1]] +
      (-C11 x2 y1 + C21 xx y1 + C12 x2 yy - C22 xx yy)/((x1 - x2) (y1 - y2));
    Ξ[[ξ, η - ξ + 1, k + 1, l]] = Ξ[[ξ, η - ξ + 1, k + 1, l]] +
      (-C11 x1 y2 + C21 xx y2 + C12 x1 yy - C22 xx yy)/((x1 - x2) (y1 - y2));
```

```
Ξ[[ξ, η − ξ + 1, k + 1, l + 1]] = Ξ[[ξ, η − ξ + 1, k + 1, l + 1]] +
                  C11 x1 y1 − C21 xx y1 − C12 x1 yy + C22 xx yy
                  ----------------------------------------------;),
                                (x1 − x2) (y1 − y2)
        {l, n + 2}]), {k, m + 2}];), {ξ, 1, m}, {η, ξ, n}];]
```

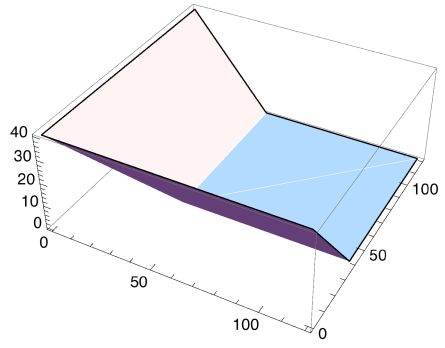
Out[111]= {391.51, Null}

Now we can turn to some concrete applications. First, we will consider an American put option with strike price 40 on the choice of one of two uncorrelated assets (the owner of the option can choose one of the two assets when the option is exercised). The termination payoff from this option is

In[112]:=
```
Λ[x_, y_] = Max[40 − Min[x, y], 0];
```

In[113]:=
```
Plot3D[Λ[x, y], {x, 0, 120}, {y, 0, 120},
  PlotPoints → 250, Mesh → False, PlotRange → All]
```

Out[113]=



The two underlying assets are uncorrelated and follow the processes $x_0\, e^{\sigma W_t}$, $t \geq 0$, and $y_0\, e^{\sigma B_t}$, $t \geq 0$, where B and W are the Brownian motions described earlier in this section. Let $(x, y) \longrightarrow f_t(x, y)$ be the value function with t years left to expiry, that is, if the time left to expiry is t years and the prices of the two assets are, respectively, $x \in \mathbb{R}_+$ and $y \in \mathbb{R}_+$, then the value of the option is $f_t(x, y) \in \mathbb{R}_+$ and we remark that what is meant here as "the value of the option" is actually a function defined on the entire range of prices, that is, the range of prices covered by the interpolation grid. Clearly, we must have $f_t(x, 0) = f_t(0, y) = 40$. Furthermore, when one of the assets has a very large value, then the option is very close to a canonical American put option on the other asset. Consequently, the values at the grid points $(\overline{x}_k, \overline{y}_l)$, where $k = 1$ or $l = 1$, will never be updated and will remain forever fixed at the initial value 40. When k is fixed to either $k = m + 2$ or $k = m + 3$, the values at the grid points $(\overline{x}_k, \overline{y}_l)$, $1 \leq l \leq n + 3$, will be updated exactly as we did earlier in the case of an American put on a single asset with a strike price 40. Similarly, when l is fixed to either $l = n + 2$ or $l = m + 3$, the values at the grid points $(\overline{x}_k, \overline{y}_l)$, $1 \leq k \leq m + 3$, will be updated in the same way, that is, as if we were dealing with an American put on a single asset with strike price 40. Of course, since the payoff is symmetric

and so is the density $p(\cdot, \cdot)$, we only need to update the values at the grid points $(\overline{x}_k, \overline{y}_l)$ for $k = 1, \ldots, m+3$ and $l = k, \ldots, n+3$.

Now we turn to the actual calculation. By using the method of dynamic integration of interpolating functions, we will compute—approximately—the pricing function $(x, y) \longrightarrow f_1(x, y)$ that maps the prices of the underlying assets in the range covered by the grid into the price of the option with one year left to maturity. With time step $t = 0.05$, we need to perform 20 iterations in the dynamic integration procedure. Note that if we choose to perform the first iteration by direct numerical integration of the termination payoff, then that would require the calculation of 741 integrals.

We initialize the procedure by tabulating the termination payoff over the interpolation grid.

```
In[114]:= Timing[V3 = Table[Λ[ XX[[k]],  YY[[l]]], {k, 1, m + 3}, {l, 1, n + 3}];]
Out[114]= {0.0069, Null}

In[115]:= K = V3; VL = K;
```

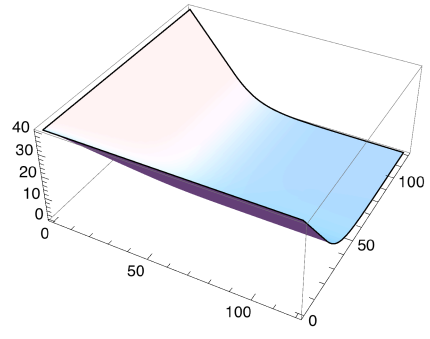
Then we do 20 iterations—at each iteration we update the tensor `VL`:

```
In[116]:= Timing[
            Do[(Do[(VL[[ξ + 1, η + 1]] = {Max[Λ[ XX[[ξ + 1]],  YY[[η + 1]]], Flatten[
                                Ξ[[ξ, η − ξ + 1]].Flatten[K]]}), {ξ, m}, {η, ξ, n}];
              Do[VL[[k + 1, n + 2]] = {Max[Max[40 − XX[[k + 1]], 0],
                    Flatten[K[[All, n + 2]].ω[[k]]]}, {k, m}];
              Do[VL[[k + 1, n + 3]] = {Max[Max[40 − XX[[k + 1]], 0],
                    Flatten[K[[All, n + 3]].ω[[k]]]}, {k, m}];
              Do[(VL[[ξ, η]] = VL[[η, ξ]]), {ξ, 2, m + 3}, {η, ξ − 1}];
              K = VL), {20}];]
Out[116]= {3.61598, Null}
```

Now we can produce the pricing map with one year left to maturity.

```
In[117]:= floc = ListInterpolation[
                Table[{ {XX[[k]],  YY[[l]]}, {VL[[k, l]]}}, {k, 1, m + 3}, {l, 1, n + 3}]];

In[118]:= Plot3D[floc[x, y], {x, 0, 120}, {y, 0, 120},
            PlotPoints → 250, Mesh → False, PlotRange → All]
```

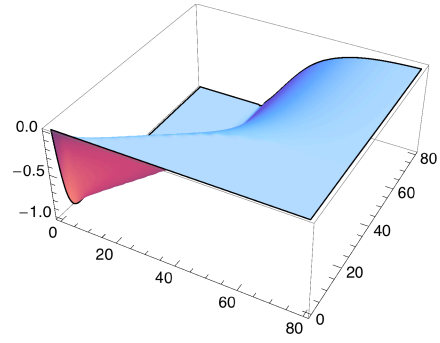
Out[118]=



Calculating the derivatives of the value function (treated as functions, too) is straightforward as we now illustrate, and before we do we remark that in the realm of finance, information about these derivatives (known as the deltas of the option) is often more important than the pricing function itself.

```
In[119]:= Plot3D[floc^(1,0)(x, y), {x, 0, 80}, {y, 0, 80},
            PlotPoints → 250, Mesh → False, PlotRange → All]
```

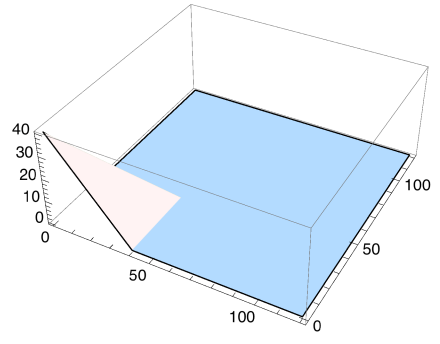
Out[119]=



In our second example, we consider an American put option on the *more expensive* of two given underlying assets (in contrast, in our previous example we dealt with an American put option on the less expensive of the two underlying assets). In this case, the termination payoff function is given by

```
In[120]:= Λ[x_, y_] = Max[40 − Max[x, y], 0];

In[121]:= Plot3D[Λ[x, y], {x, 0, 120}, {y, 0, 120},
            PlotPoints → 250, Mesh → False, PlotRange → All]
```

Out[121]=

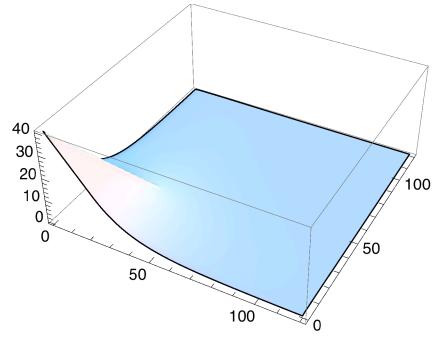


This time the boundary conditions must take into account the fact that when one of the assets becomes too expensive, the option becomes worthless, and when one of the assets is worthless, the option may be treated as a standard American option on the other asset. For the purpose of illustration we will compute the value of the option with three years left to maturity.

```
In[122]:= Timing[V4 = Table[Λ[XX[[k]], YY[[l]]], {k, 1, m + 3}, {l, 1, n + 3}];]

Out[122]= {0.007632, Null}

In[123]:= K = V4; VL = K;
```

```
In[124]:= Timing[
  Do[(Do[(VL[[ξ + 1, η + 1]] = {Max[Λ[ XX[[ξ + 1]],  YY[[η + 1]]], Flatten[
                  Ξ[[ξ, η − ξ + 1]].Flatten[K]]}), {ξ, m}, {η, ξ, n}];
    Do[VL[[1, k + 1]] = {Max[Max[40 −  YY[[k + 1]], 0],
        Flatten[K[[1, All]]].ω[[k]]]}, {k, m}]; Do[
     VL[[k + 1, n + 2]] = {0}, {k, m}]; Do[VL[[k + 1, n + 3]] = {0}, {k, m}];
    Do[(VL[[ξ, η]] = VL[[η, ξ]]), {ξ, 2, m + 3}, {η, ξ − 1}]; K = VL), {60}];]

Out[124]= {12.2292, Null}

In[125]:= floc = ListInterpolation[
    Table[{{XX[[k]],  YY[[l]]}, {VL[[k, l]]}}, {k, 1, m + 3}, {l, 1, n + 3}]];

In[126]:= Plot3D[floc[x, y], {x, 0, 120}, {y, 0, 120},
   PlotPoints → 250, Mesh → False, PlotRange → All]
```

Out[126]=

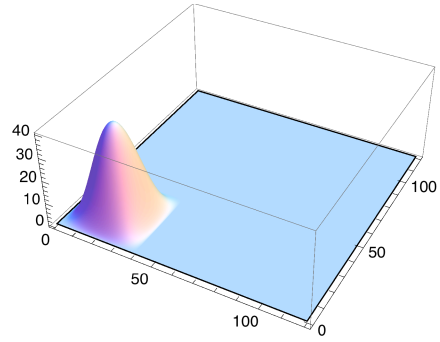


This is a good example of a value function which is neither everywhere smooth nor convex.

In our final example we consider optimal stopping of the same diffusion in $\mathbb{R}^2$ but with smooth termination payoff of the form

```
In[127]:= λ[x_] = 1/100 x^2 * 1/40 (Max[40 − x, 0])^2; Λ[x_, y_] = 1/40 λ[x] * λ[y];

In[128]:= Plot3D[Λ[x, y], {x, 0, 120}, {y, 0, 120},
   PlotPoints → 250, Mesh → False, PlotRange → All]
```

Out[128]=

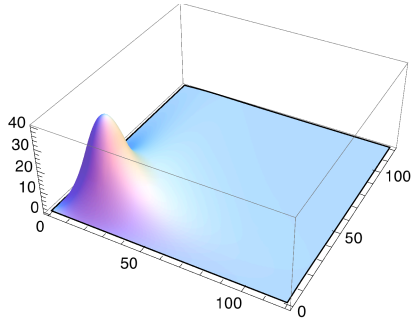


With this termination payoff the boundary conditions must reflect the fact that the option is worthless when one of the assets is worthless or when one of the

assets is too expensive. We will execute 60 iterations, which will give us the value function with three years left to expiry.

```
In[129]:= Timing[V5 = Table[Λ[XX[[k]], YY[[l]]], {k, 1, m + 3}, {l, 1, n + 3}];]

Out[129]= {0.015148, Null}

In[130]:= K = V5; VL = K;

In[131]:= Timing[
            Do[(Do[(VL[[ξ + 1, η + 1]] = {Max[Λ[XX[[ξ + 1]], YY[[η + 1]]], Flatten[
                                Ξ[[ξ, η − ξ + 1]].Flatten[K]]}), {ξ, m}, {η, ξ, n}];
              Do[VL[[1, k + 1]] = {0}, {k, m}]; Do[VL[[k + 1, n + 2]] = {0}, {k, m}];
              Do[VL[[k + 1, n + 3]] = {0}, {k, m}];
              Do[(VL[[ξ, η]] = VL[[η, ξ]]), {ξ, 2, m + 3}, {η, ξ − 1}]; K = VL), {60}];]

Out[131]= {12.3947, Null}

In[132]:= floc = ListInterpolation[
              Table[{{XX[[k]], YY[[l]]}, {VL[[k, l]]}}, {k, 1, m + 3}, {l, 1, n + 3}]];

In[133]:= Plot3D[floc[x, y], {x, 0, 120}, {y, 0, 120},
            PlotPoints → 250, Mesh → False, PlotRange → All]
```

Out[133]=



■ Conclusions

Most man-made computing devices can operate only with finite lists of numbers or symbols. Any use of such devices for modeling, analysis, and optimal control of stochastic systems inevitably involves the encoding of inherently complex phenomena in terms of finite lists of numbers or symbols. Thus one can think of two general directions that may lead to expanding the realm of computable models. First, one may try to construct computing devices that can handle larger and larger lists, faster and faster. Second, one may try to develop "smarter" procedures with which more complex objects can be encrypted with shorter lists of numbers. With this general direction in mind, the methodology developed in this article is entirely logical and natural. Indeed, the approximation of functions with splines or other types of interpolating functions is a familiar, well developed, and entirely natural computing tool. The use of special quadrature/cubature rules—as opposed to general methods for numerical integration—in the context of dynamic integration of interpolating functions is

just as logical and natural. However, only recently have computer languages in which one can implement such procedures in a relatively simple and straightforward fashion become widely available. This article is an attempt to demonstrate how the advent of more sophisticated computing technologies may lead to the development of new and more efficient algorithms. Curiously, new and more efficient algorithms often lead to the design of new and more efficient computing devices. In particular, all procedures described in this article can be implemented on parallel processors or on computing grids in essentially a trivial way. There is a strong incentive to build computing devices that can perform simultaneously several numerical integrations, or can compute dot products between very large lists of numbers very fast.

Finally, we must point out that most of the examples presented in the article are only prototypes. They were meant to be executed on a generic (and slightly out of date) laptop computer with the smallest possible number of complications. Many further improvements in terms of higher accuracy and overall efficiency can certainly be made.

■ Acknowledgments

The author thanks all three anonymous referees for their extremely helpful comments and suggestions.

■ References

[1] A. Lyasoff, "Path Integral Methods for Parabolic Partial Differential Equations with Examples from Computational Finance," *The Mathematica Journal*, **9**(2), 2004 pp. 399-422.

[2] A. Bensoussan and J. L. Lions, *Applications of the Variational Inequalities in Stochastic Control*, Amsterdam: North Holland, 1982.

[3] M. H. A. Davis, "Markov Models and Optimization," Monographs on Statistics and Applied Probability, Vol. 49, Chapman & Hall/CRC Press,1993.

[4] V. Bally and G. Pagès, "A Quantization Algorithm for Solving Discrete Time Multidimensional Optimal Stopping Problems," *Bernoulli*, **9**(6), 2003 pp. 1003-1049.

[5] V. Bally and G. Pagès, "Error Analysis of the Quantization Algorithm for Obstacle Problems," *Stochastic Processes and their Applications*, **106**(1), 2003 pp. 1-40.

[6] B. Bouchard, I. Ekeland, and N. Touzi, "On the Malliavin Approach to Monte Carlo Approximation of Conditional Expectations," *Finance and Stochastics*, **8**(1), 2004 pp. 45-71.

[7] M. Broadie and P. Glasserman, "Pricing American-Style Securities Using Simulation," *Journal of Economic Dynamics and Control*, **21**(8-9),1997 pp.1323-1352.

[8] F. A. Longstaff and R. S. Schwartz, "Valuing American Options By Simulation: A Simple Least-Square Approach," *Review of Financial Studies*, **14**(1), 2001 pp. 113-147.

[9] W. H. Press, B. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed., Cambridge: Cambridge University Press, 1992.

[10] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Englewood Cliffs, NJ: Prentice Hall, 1977.

[11] T. Lyons and N. Victoir, "Cubature on Wiener Space," in *Proceedings of the Royal Society of London*, Series A (*Mathematica*l and Physical Sciences), **460**(2041), 2004 pp. 169-198.

[12] S. Kusuoka, "Approximation of Expectation of Diffusion Process and Mathematical Finance," in *Advanced Studies in Pure Mathematics*, *Proceedings of the Taniguchi Conference on Mathematics (Nara 1998)*, (M. Maruyama and T. Sunada, eds.), **31**, 2001 pp. 147-165.

[13] S. Kusuoka, "Approximation of Expectation of Diffusion Processes Based on Lie Algebra and Malliavin Calculus," *Advances in Mathematical Economics*, **6**, 2004 pp. 69-83.

[14] A. H. Stroud, *Approximate Calculation of Multiple Integrals*, Englewood Cliffs, NJ: Prentice-Hall, 1971.

A. Lyasoff, "Dynamic Integration of Interpolating Functions and Some Concrete Optimal Stopping Problems," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.10.4–3.

■ Additional Material

Lyasoff.zip contains:
integrals.mx
integrals.txt
CoeffList.mx
CoeffList.txt
Xi.mx
Xi.txt

Available at www.mathematica-journal.com/data/uploads/2011/12/Lyasoff.zip.

About the Author

Andrew Lyasoff is director of the Graduate Program in Mathematical Finance at Boston University. His research interests are mainly in the areas of Stochastic Analysis, Optimization Theory, and Mathematical Finance and Economics.

Andrew Lyasoff
Boston University
Mathematical Finance Program
Boston, MA
alyasoff@bu.edu