Numerical Methods for Option Pricing under Jump-Diffusion Models

WU, Tao

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Mathematics

The Chinese University of Hong Kong

July 2010

Abstract

The classical Black-Scholes model is known for its shortcomings in modeling empirical price dynamics in daily markets, which reveals volatility smiles and heavy tails. In the recent decade, the jump-diffusion models have arisen as remedies for Black-Scholes. The present work concerns numerical computation of option prices under jump-diffusion models, namely numerical solution to partial integro-differential equations and Monte Carlo method.

In the first part of the thesis, we propose a novel parallel Talbot method (PTM) for solving jump-diffusion equations on option prices. After using standard spatial discretization, we represent the formal solution to the semidiscretized problem as a summation of φ-functions. To numerically approximate the matrix-valued φ-functions, we construct a Talbot quadrature based on the Dunford-Cauchy integral. Moreover, we derive strategy for optimal parameterization of parabolic Talbot contours. The above PTM yields a geometric convergence in temporal direction, and therefore outperforms traditional time-marching methods. We report the effectiveness of PTM in the numerical experiments, particularly in a parallel computing environment.

In the second part, we use the least squares method (LSM) to price American options under Lévy models. Essentially, LSM estimates the continuation value for each simulated path by least squares regression, and makes backward(-in-time) induction on exercising the option or not. The drawback of LSM in implementation is that its memory requirement grows like $O(mn)$, where m is the number of time steps and n is the number of simulated paths. We propose a new simulation method with memory requirement $O(m+n)$. The total computational cost is always less than twice of that of the traditional method. In the numerical experiments, we illustrate the efficiency of our method by pricing American options under several typical Lévy models.

摘 要

由於日常的市場價格呈現出“波動微笑”和“長尾”等特征，經典的Black-Scholes模型在模擬市場價格時存在缺陷。在最近的十年裡，“跳躍擴散模型”被提出以彌補Black-Scholes模型的缺陷。這篇論文關注的是基於跳躍擴散模型的期權定價中的數值計算。具體來說，這裡的數值計算方法有偏微分方程數值計算和蒙特卡洛模擬。

在第一部分的論文中，我們提出新的“並行Talbot方法”來計算期權價格的跳躍擴散方程。在標准的空間離散後，我們可以把方程的解在形式上表示為一系列“φ函數”的和。為了對這樣的形式解作數值逼近，我們基於Dunford-Cauchy積分構造了並行Talbot方法。另外，我們還提出了對拋物Talbot路徑的最優參數化方案。以上的並行Talbot方法在時間方向上可以達到幾何收斂，因此優於傳統的時間推進方法。在數值實驗中，我們記錄了並行Talbot方法的有效性，尤其在考慮並行計算的情況下。

在第二部分中，我們使用“最小二乘法”來計算Levy模型下的美式期權價格。從本質上講，最小二乘法通過最小二乘回歸來估計每條模擬路徑上的持有價值，然後（在時間上）向後歸納得出是否行使期權。最小二乘法在算法實現中的缺點是它對內存需求的增長為O(mn)。這裡m是時間步數，而n是模擬路徑數。我們提出了新的模擬方法，使得內存需求減少為O(m+n)的，而總計算量被控制在小於兩倍的傳統方法。在數值實驗中，我們展示了新方法可以有效處理幾種典型Levy模型下的美式期權定價問題。

Acknowledgements

I am most grateful to my supervisor Prof. Raymond Chan for creating me an inspiring academic environment for the past two years. His enthusiasm and vision in mathematics enlightens me and other people around him. In particular, I would like to thank him for introducing me the subject of this thesis and for his motivation and encourage throughout my thesis work.

I would also like to thank all faculty members at our math department, for their devotion in teaching and their sharing of expertise during my six years at the Chinese University.

My special gratitude goes to Prof. I-Liang Chern for his endless insights, and to Prof. Michael Hintermüller for his influence on my future research interests.

The technical support from Frank Ng and Stephan Chan on parallel computing is also gratefully acknowledged.

Finally, I thank all TOTS teammates for their support all the time. I am proud to be part of you.

Contents

Chapter 1

Background and Organization

Ever since the seminal work by Black and Scholes in 1973 [8], the principle of arbitrage-free (or risk-neutral) pricing for financial derivatives has become the cornerstone in the field of mathematical finance. Their work was later connected and expanded with the theory of martingale, first due to Harrison and Pliska [25]. We refer to standard textbooks [58, 7, 50, 19] for more background on risk-neutral option pricing.

Briefly speaking, in a market consisting of risky assets and options written on those assets, the market is arbitrage-free if and only if there is a risk-neutral probability measure equivalent to the physical probability measure, see e.g. [19, Theorem 3.5.1]. Suppose such a risk-neutral measure exists, then the arbitrage-free price (or the risk-neutral price) of an option is given by the discounted expected payoff under a (generally nonunique) risk-neutral measure, see e.g. [19, Theorem 4.5.1]. The present thesis falls into the general framework of arbitrage-free pricing.

In the thesis, we consider jump-diffusion models that have been popularized in financial modeling during the past decade, cf. [14]. The classical Black-Scholes model [8] is criticized for its oversimplification, as the asset price in the Black-Scholes model follows a geometric Brownian motion. However, the empirical observation in real financial trading reveals that the implied volatility surface often displays a volatility smile [36]. Moreover, the distribution of the asset return, assumed to be Gaussian in the Black-Scholes model, exhibits a heavy tail [13], i.e. large moves of the market have decent probabilities to occur. As remedies for Black-Scholes, the jump-diffusion models (or more generally the Lévy models) contain discontinuous jumps in addition to the classical diffusion, so that the phenomena of the volatility smiles and the heavy tails can be generically accounted for [14].

It is worth noting that jump-diffusion models are (linear) models for an incomplete market, where no unique risk-neutral measure (or no perfect hedging strategy) exists. We refer to the reference [14] for further background on modeling aspects of jump-diffusion models, such as how to choose the pricing risk-neutral measure under jump-diffusion models (so called "model calibration").

Along with the new models come the new challenges on numerical (pricing) aspects. The present thesis concerns numerical computation of option prices under jump-diffusion models. The rest of the thesis divides into two major methodologies: numerical solution to partial (integro-)differential equations for Chapter 2 and Monte Carlo simulation for Chapter 3. Each chapter is self-contained, with introduction, problem formulation, methodology, numerical experiments, and conclusion therein.

Chapter 2

Parallel Talbot method for solving partial integro-differential equations

2.1 Introduction

The option price, where the underlying asset price follows jump-diffusion processes, is governed by partial integro-differential equation (PIDE) [14]. The valuation of option prices via numerically solving PIDE has been considered by many authors [5, 3, 15, 18, 37, 52, 20]. However, most authors consider the method of lines, yielding a polynomial temporal convergence rate (usually up to 2nd-order) due to stability constraints from the stiffness. Andersen and Andreasen [5] propose an alternating direction implicit (ADI) operator splitting method with a 2nd-order temporal accuracy; Almendral and Oosterlee [3] use a 2nd-order backward differentiation formula (BDF2); Cont and Voltchkova [15] consider the implicit-explicit (IMEX) scheme, i.e. implicit for the differential operator (stiff part) and explicit for the integral operator (nonstiff part), with 1st-order accuracy; d'Halluin, Forsyth and Vetzal [18] consider Crank-Nilcoson time marching scheme with 2nd-order accuracy; Toivanen [52] performs the implicit Rannacher time stepping with nonuniform grids; Feng and Linetsky [20] accelerate the IMEX scheme with Richardson extrapolation, attaining a temporal accuracy with arbitrary order.

Since the pioneering work by Talbot [51], the time discretization, based on contour integration and the consequent quadrature rule, has been proposed and developed for parabolic problems, see e.g. [48, 49, 21, 33, 22, 31, 34, 38, 54, 55, 23, 57, 56]. After spatial discretization, the initial-boundary value problem (IBVP) is reduced to the semidiscrete equation. Further after time integration, one is left to evaluate the solution formula consisting of so-called φ-functions. These Talbot-type methods numerically approximate the φ-functions based on the Dunford-Cauchy integral representation with deformed Talbot contour (often parameterized as a parabola, a hyperbola, or a cotangent contour). Then one discretizes the integral with trapezoidal rule and truncates the infinite series. This process, named *Talbot quadrature*, implicitly constructs a rational approximation, which attains a geometric convergence rate. A promising feature of the Talbot-type methods lies in its two levels of parallelism, i.e. with respect to various time points and with respect to the summands in the Talbot quadrature formula. For this reason, we refer to this approach as "parallel Talbot method" (PTM) in the present work. Notably, the performance of the PTM can be much improved by optimized parameterization for the aforementioned contours, cf. [55, 34, 57]. Thus, we see potential applications of PTM for high-performance computing purpose, e.g. in option pricing.

Surprisingly, such applications are not yet widely seen in the literature. The obstacles are the followings. First of all, the problems from applications, such as the semidiscretized jump-diffusion equation, are often nonnormal. In this scenario, the contour in [57] could break down, see e.g. [56] for convection-diffusion problems. Secondly, although the Talbot contours for sectorial operators are considered for nonnormal operators, see e.g. [49, 31, 34], the translation from the assumptions to the specific problems remains largely unclear. We aim to bridge the gap by applying PTM to the nonnormal jump-diffusion problems in finance. We control the (pseudo)spectrum of the nonnormal jump-diffusion operator by a parabola, based on which we propose a parabolic contour with optimal parameterization. In addition, our proposed Talbot quadrature works for general φ-functions using so-called common poles, which greatly reduces the computational cost. In the numerical experiments, we verify the optimality of the proposed contour and show the competiveness of PTM against the extrapolation IMEX method (ext-IMEX) on parallel machine.

The remaining sections in this chapter are organized as follows. We formulate the initial-boundary value problem for option pricing in Section 2, and semidiscretize the problem in Section 3. We present PTM in details in Section 4. The numerical results are provided in Section 5. Section 6 concludes the chapter.

2.2 Initial-boundary value problem

Under a given risk-neutral measure, the price of a stock is modeled as $S_t = Ke^{X_t}$ such that $\mathbb{E}[S_t|S_0] = S_0e^{(r-q)t}$, where S_0 is the current stock price, K is the strike price at maturity, and $(X_t)_{t\geq 0}$ is a (finite-activity) jump-diffusion process [14] satisfying $X_0 = \log(S_0/K)$ and at time $t > 0$,

$$dX_t = (r - q - \frac{1}{2}\sigma^2 - \lambda\kappa)dt + \sigma dW_t + d\left(\sum_{n=1}^{N_t} J_n\right).$$

Here $r \geq 0$ is the risk-free interest rate, $q \geq 0$ is the continuous dividend yield, $\sigma > 0$ is the stock return volatility, W_t is a standard Brownian motion, N_t is a Poisson process with intensity rate $\lambda > 0$, $(J_n)_{n=1}^{N_t}$ is a sequence of independent identically distributed random variables from a given distribution $f(x)$, and $\kappa = \int_{\mathbb{R}}(e^x - 1)f(x)dx$ is the compensation rate of a Poisson jump.

As in [3, 15, 18, 52], the value of an option $u(t,x)$ on the stock can be obtained by solving the initial-boundary value problem (IBVP) on the computational domain $[0,T]\times\Omega$

$$\frac{\partial u}{\partial\tau}=a_2\frac{\partial^2 u}{\partial x^2}+a_1\frac{\partial u}{\partial x}+a_0u+\lambda\int_{\mathbb{R}}u(\tau,x)f(x-y)dy,\quad (\tau,x)\in(0,T]\times\Omega; \tag{2.1}$$

$$u(0,x)=\psi(x),\ x\in\mathbb{R};\quad u(\tau,x)=R(x),\ (\tau,x)\in[0,T]\times(\mathbb{R}\backslash\Omega). \tag{2.2}$$

Here $\tau=T-t$, $a_2=\sigma^2/2$, $a_1=r-q-\sigma^2/2-\lambda\kappa$, $a_0=-r-\lambda$, $\psi(x)$ is the payoff function, e.g. $\psi(x)=\max(Ke^x-K,0)$ for a call, and the rebate function $R(x)$ is imposed wherever $x\in\mathbb{R}\backslash\Omega$.

Note that the IBVP (2.1)–(2.2) is a *localized* problem, as we only need to solve $u(\tau,x)$ on the bounded time-space domain $[0,T]\times\Omega$. This IBVP framework is robust in handling European options and exotic barrier options. For a European option, we introduce a change of variable $\widetilde{u}(\tau,x)=e^{r\tau}u(\tau,x-r\tau)$, and then solve (2.1)–(2.2) with modified coefficients $\widetilde{a_1}=a_1-r$, $\widetilde{a_0}=a_0+r$. For computation, we take $\Omega:=[\underline{x},\overline{x}]$ to be the localization domain and impose the asymptotical boundary condition $R(x):=\psi(x)$. As $\underline{x}\to-\infty$ and $\overline{x}\to\infty$, the localization error decays exponentially with the size of the domain Ω, cf. [15]. For another example, consider an up-and-out barrier option with upper barrier U. Different from the European option, we do not need the change of variable for u. Besides, we take $\overline{x}=U$, $R(x)=0$ for $x\geq U$, and $R(x)=\psi(x)$ for $x<U$. We remark that with appropriate change of variable, we have that R is τ-independent.

2.3 Spatial discretization and semidiscrete problem

We discretize the PIDE in space by finite difference as considered in [3, 15, 18, 52]. Let $x_{\min}=x_0<x_1<...<x_M=x_{\max}$, $\Delta x=(x_{\max}-x_{\min})/M$, $\Omega=[\underline{x},\overline{x}]\triangleq[x_{\min}+\Delta x/2,\ x_{\max}-\Delta x/2]$, and $u(\tau,x)$ be semidiscretized into the vector $\mathbf{u}(\tau)$

with $(\mathbf{u}(\tau))_j \equiv u(\tau, x_j)$. In the following, we shall omit "$(\cdot)$" after u and $\mathbf{u}$ given no confusion.

For the differential operators, we use the 2nd-order finite difference

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_j \approx \frac{\mathbf{u}_{j+1} - 2\mathbf{u}_j + \mathbf{u}_{j-1}}{(\Delta x)^2}, \quad \left(\frac{\partial u}{\partial x}\right)_j \approx \frac{\mathbf{u}_{j+1} - \mathbf{u}_{j-1}}{2\Delta x}.$$

For the integral operator, we split the integral into two parts

$$\int_{\mathbb{R}} (\cdot) dy = \int_{\Omega} (\cdot) dy + \int_{\mathbb{R}\setminus\Omega} (\cdot) dy.$$

We approximate the integral over Ω with a trapezoidal rule

$$\int_{\Omega} u(\tau, y) f(y - x_j) dy \approx \Delta x \sum_{k=1}^{M-1} \mathbf{u}_k f(x_k - x_j).$$

In matrix form, we write

$$\mathbf{D} = \text{tridiag}\left[\frac{a_2}{(\Delta x)^2} + \frac{a_1}{\Delta x}, -\frac{2a_2}{(\Delta x)^2} - r - \lambda, \frac{a_2}{(\Delta x)^2} - \frac{a_1}{\Delta x}\right], \tag{2.3}$$

$$\mathbf{J} = \lambda \Delta x \begin{bmatrix} f(0) & f(\Delta x) & \cdots & f((M-2)\Delta x) \\ f(-\Delta x) & \ddots & \ddots & \vdots \\ \vdots & \ddots & & \vdots \\ f(-(M-2)\Delta x) & \cdots & \cdots & f(0) \end{bmatrix}, \tag{2.4}$$

and let $\mathbf{A} := \mathbf{D} + \mathbf{J}$.

Assumption 2.3.1. *We use the following assumptions as in [3, 18] that*

(1) $r \geq 0$*;*

(2) $a_2 > |a_1 \Delta x|$*;*

(3) $\max_j \left(\sum_k \mathbf{J}_{jk}\right) \leq \lambda$.

Note that in practice the mesh size Δx is often small so that the condition 2.3.1(2) holds. Beside, since $\sum_k \mathbf{J}_{jk}$ is the trapezoidal approximation of an integral of the probability density $f(x)$ over a truncated domain, the condition

2.3.1(3) also holds. With Assumption 2.3.1, the matrix $-\mathbf{D} - \lambda\mathbf{I}$ is diagonally dominant M-matrix, and that the matrix $-\mathbf{J} + \lambda\mathbf{I}$ is a diagonally dominant M-matrix. Therefore, the matrix $-\mathbf{A}$ is a diagonally dominant *M-matrix* [6], i.e.

$$\sum_k \mathbf{A}_{jk} \leq 0,\ \forall j; \quad \mathbf{A}_{jk} > 0,\ \forall j \neq k.$$

For the integral over $\mathbb{R}\backslash\Omega$, denoted by $\xi(x)$, we plug in the boundary condition

$$\xi(x) = \lambda \int_{\mathbb{R}\backslash\Omega} R(y) f(y-x) dy, \tag{2.5}$$

which can be calculated analytically or numerically [3]. This is referred to as the *truncation* of large jumps, according to Cont and Voltchkova [15].

Then we combine all the boundary terms into the vector $\mathbf{b}$, i.e.

$$\mathbf{b}_j = \begin{cases} \xi(x_j) + R(x_{\min})[a_2(\Delta x)^{-2} - a_1(\Delta x)^{-1}], & \text{if } j = 0; \\ \xi(x_j) + R(x_{\max})[a_2(\Delta x)^{-2} + a_1(\Delta x)^{-1}], & \text{if } j = M; \\ \xi(x_j), & \text{otherwise.} \end{cases}$$

Thus, we are left with the semidiscretized problem

$$\frac{\partial}{\partial \tau}\mathbf{u}(\tau) = \mathbf{A}\mathbf{u}(\tau) + \mathbf{b},\ 0 < \tau \leq T; \quad (\mathbf{u}(0))_j = \psi(x_j). \tag{2.6}$$

By performing the time integration on (2.6), we have the formal solution

$$\mathbf{u}(T) = \varphi_0(\mathbf{A})\mathbf{u}(0) + \varphi_1(\mathbf{A})\mathbf{b}. \tag{2.7}$$

The functions φ_0 and φ_1 are defined by

$$\varphi_0(\mathbf{A}) = e^{T\mathbf{A}}, \quad \varphi_1(\mathbf{A}) = \mathbf{A}^{-1}(e^{T\mathbf{A}} - \mathbf{I}). \tag{2.8}$$

It is the goal of Section 2.4 to develop an efficient method to evaluate the solution formula (2.7), or so-called *φ-functions.*

2.4 Parallel Talbot method

2.4.1 φ-functions and Talbot quadrature

The family of φ-functions arises from the exponential time differencing methods for first-order semilinear problems [16, 29, 40]. The φ-functions, with scalar arguments, can be defined by the integral representation

$$\varphi_l(z) = \int_0^T e^{(T-s)z} \frac{s^{l-1}}{(l-1)!} ds, \quad \text{for } l \geq 1; \quad \varphi_0(z) = e^{Tz}, \tag{2.9}$$

or defined by the recurrence

$$\varphi_l(z) = \left. \frac{\varphi_{l-1}(z) - T^{l-1}/(l-1)!}{z} \right|_{z \neq 0}, \ \varphi_l(0) = \frac{T^l}{l!}, \text{ for } l \geq 1; \quad \varphi_0(z) = e^{Tz}. \tag{2.10}$$

Observe that these φ-functions can be regarded as the regular part of the Laurent series of e^{Tz}/z^l

$$\frac{e^{Tz}}{z^l} = \varphi_l(z) + \sum_{k=0}^{l-1} \frac{T^k}{k!} z^{k-l}. \tag{2.11}$$

The formulae (2.9)–(2.11) for the scalar-valued φ-functions extend naturally to matrix-valued cases using the Jordan canonical form, cf. [28, 26].

We begin by evaluating $\varphi_0(\mathbf{A})\mathbf{v}$ for a given vector $\mathbf{v}$. The evaluation is based on the Dunford-Cauchy integral

$$\varphi_0(\mathbf{A})\mathbf{v} = \int_\Gamma \frac{1}{2\pi i} e^{Tz} (z\mathbf{I} - \mathbf{A})^{-1} \mathbf{v} \, dz, \tag{2.12}$$

where Γ is a *Talbot contour* encircling the origin and the spectrum of $\mathbf{A}$ in anticlockwise sense. These Talbot-type methods are effective mainly because the integrand in (2.12) decays exponentially fast as $\operatorname{Re} z \to -\infty$. In the following, we describe how to construct the Talbot quadrature based on the contour integral (2.12).

We parameterize the contour $\Gamma : \theta \mapsto z(\theta)$, and rewrite the formula (2.12) as

$$\varphi_0(\mathbf{A})\mathbf{v} = \int_{-\infty}^{\infty} \frac{1}{2\pi i} e^{Tz(\theta)} [z(\theta)\mathbf{I} - \mathbf{A}]^{-1} z'(\theta) \mathbf{v} \, d\theta. \tag{2.13}$$

Then we apply a midpoint rule to the integral on an equispaced grid $\theta_j := (j + 1/2)h,\ j \in \mathbb{Z}$, i.e.

$$\varphi_0(\mathbf{A})\mathbf{v} \approx h \sum_{j=-\infty}^{\infty} \frac{1}{2\pi i} e^{Tz(\theta_j)} [z(\theta_j)\mathbf{I} - \mathbf{A}]^{-1} z'(\theta_j)\mathbf{v}. \tag{2.14}$$

Finally we truncate the infinite sum

$$\varphi_0(\mathbf{A})\mathbf{v} \approx h \sum_{j=-N}^{N-1} \frac{1}{2\pi i} e^{Tz(\theta_j)} [z(\theta_j)\mathbf{I} - \mathbf{A}]^{-1} z'(\theta_j)\mathbf{v}. \tag{2.15}$$

Observe that the Talbot quadrature (2.15) implicitly constructs a rational approximation with poles $(z_j)_{j=-N}^{N-1}$ and weights $\left(c_j^{(0)}\right)_{j=-N}^{N-1}$

$$\varphi_0(\mathbf{A})\mathbf{v} \approx h \sum_{j=-N}^{N-1} c_j^{(0)} (z_j\mathbf{I} - \mathbf{A})^{-1}\mathbf{v},$$

where

$$z_j := z(\theta_j), \quad c_j^{(0)} := \frac{e^{Tz_j} z'(\theta_j)}{2\pi i}.$$

Now we turn to the evaluation of general $\varphi_l(\mathbf{A})\mathbf{v}$ for $l \geq 1$. It leads to poor convergence rate if we directly apply the Talbot quadrature. This is because the functions $(\varphi_l(z))_{l\geq 1}$ are known to be of algebraic decay as Re $z \to -\infty$. More precisely, this algebraic decay is due to the principal part (or the summation on right hand side) in (2.11). Fortunately, we are allowed to ignore this term, guaranteed by the following theorem.

Theorem 2.4.1. *[47, Theorem 4.2.4] Suppose that a simple contour Γ encircles the origin and the spectrum* $\mathbf{A}$. *Then*

$$\varphi_l(\mathbf{A})\mathbf{v} = \int_\Gamma \frac{1}{2\pi i} \frac{e^{Tz}}{z^l} (z\mathbf{I} - \mathbf{A})^{-1}\mathbf{v}\ dz. \tag{2.16}$$

Consequently, in analogy with (2.13)–(2.15) we have

$$\begin{aligned}
\varphi_l(\mathbf{A})\mathbf{v} &= \int_{-\infty}^{\infty} \frac{1}{2\pi i} \frac{e^{Tz(\theta)}}{z(\theta)^l} [z(\theta)\mathbf{I} - \mathbf{A}]^{-1} z'(\theta)\mathbf{v}\ d\theta && (2.17)\\
&\approx h \sum_{j=-\infty}^{\infty} \frac{1}{2\pi i} \frac{e^{Tz_j}}{z_j^l} (z_j\mathbf{I} - \mathbf{A})^{-1} z'(\theta_j)\mathbf{v} && (2.18)\\
&\approx h \sum_{j=-N}^{N-1} c_j^{(l)} (z_j\mathbf{I} - \mathbf{A})^{-1}\mathbf{v}, && (2.19)
\end{aligned}$$

with poles and weights given by

$$z_j := z(\theta_j), \quad c_j^{(l)} := \frac{e^{Tz_j} z'(\theta_j)}{2\pi i z_j^l}. \tag{2.20}$$

Now we see the significance of Theorem 2.4.1. Provided that

$$|z^{-l}| = O(1), \tag{2.21}$$

the Talbot quadrature with poles $(z_j)_{j=-N}^{N-1}$ (and the corresponding error estimate) applies to general φ_l functions, see (2.17)-(2.19). For this reason, we call such poles the *common poles*.

Thus, our approximation to the solution formula (2.7) becomes

$$\begin{aligned} \mathbf{u}(T) &\approx h \sum_{j=-N}^{N-1} c_j^{(0)} (z_j \mathbf{I} - \mathbf{A})^{-1} \mathbf{u}(0) + h \sum_{j=-N}^{N-1} c_j^{(1)} (z_j \mathbf{I} - \mathbf{A})^{-1} \mathbf{b} && (2.22) \\ &= h \sum_{j=-N}^{N-1} (z_j \mathbf{I} - \mathbf{A})^{-1} \left[c_j^{(0)} \mathbf{u}(0) + c_j^{(1)} \mathbf{b} \right] && (2.23) \\ &= \mathrm{Re} \left\{ 2h \sum_{j=0}^{N-1} (z_j \mathbf{I} - \mathbf{A})^{-1} \left[c_j^{(0)} \mathbf{u}(0) + c_j^{(1)} \mathbf{b} \right] \right\}. && (2.24) \end{aligned}$$

Notice that by using the common poles we reduce the computational cost by half. Due to the symmetry that $\overline{z(\theta_j)} = z(\theta_{-j-1})$ and $\overline{z'(\theta_j)} = z'(\theta_{-j-1})$, the summands in (2.23) come in conjugate pairs, leading to another reduction of cost by half. Thus, the remaining task is to solve N independent linear systems in the formula (2.24), which adopts a parallel implementation.

Nevertheless, we have not touched upon how to choose the Talbot contour Γ in (2.16) properly. This is the focus of the remaining of this chapter. In Subsection 2.4.3, we shall devise optimal strategy on how to parameterize parabolic Talbot contour. Before that we need preparation on spectrum estimation for non-normal jump-diffusion operator, which leads to feasibility constraints in contour parametrization.

2.4.2 Control on nonnormality and feasibility constraints

Weideman and Trefethen [57] devised the optimal parameterization of Talbot contour as a parabola. Yet they assume that the spectrum of **A** is restricted on the negative real axis. Such a parabolic contour can fail for nonnormal operators, e.g. in convection-diffusion problems, cf. [56], and also in jump-diffusion problems, see e.g. Figure 2.4 in Section 2.5. In this subsection, we investigate how to control the nonnormal effect, and how such control leads to (explicit) feasibility constraints in constructing parabolic Talbot contour for jump-diffusion problems.

We denote the infinitesimal generator of the jump-diffusion equation (2.1) by $\mathcal{A} := \mathcal{D} + \mathcal{J}$, where

$$\mathcal{D}u = a_2 \frac{\partial^2 u}{\partial x^2} + a_1 \frac{\partial u}{\partial x} + a_0 u, \quad \mathcal{J}u = \lambda \int_{\mathbb{R}} u(x+y) f(y) dy.$$

When acting on (a dense subspace of) $L^2(\mathbb{R})$, the operator $\mathcal{A}$ can be diagonalized by Fourier modes and is normal. However, it becomes nonnormal after being localized, truncated, and discretized into **A**, see (2.1), (2.5), and (2.3)–(2.4) respectively. Due to the nonnormality, the eigenvalues of **A** are ill-conditioned and can move off the negative real axis into the left half complex plane. See the location of the eigenvalues of **A** in Figure 2.2 for example. While the behavior of a normal operator is often completely governed by its eigenvalues, it is difficult (and meaningless) to analyze the eigenvalues of a nonnormal operator. Instead, it is more meaningful to analyze the pseudospectra for nonnormal operators, cf. [53].

To estimate the pseudospectra, we employ techniques based on the *symbol curve* and the associated *winding number*, as in [43, 44, 56, 53]. The symbol curve of $\mathcal{A}$ is given by

$$\Phi_{\mathcal{A}}(\omega) = -a_2\omega^2 + ia_1\omega + a_0 + \lambda \hat{f}(\omega),\ \omega \in \mathbb{R}, \text{ where } \hat{f}(\omega) = \int_{\mathbb{R}} f(y) e^{i\omega y} dy. \quad (2.25)$$

Note that the symbol curve $(\Phi_{\mathcal{A}}(\omega))_{\omega\in\mathbb{R}}$ is not closed. As in [53], we define the winding number $I(\Phi_{\mathcal{A}}(\mathbb{R}); z)$ as follows. Let $(\Phi_{\mathcal{A}}(\omega))_{\omega\in[-\overline{\omega},\overline{\omega}]}$ be completed

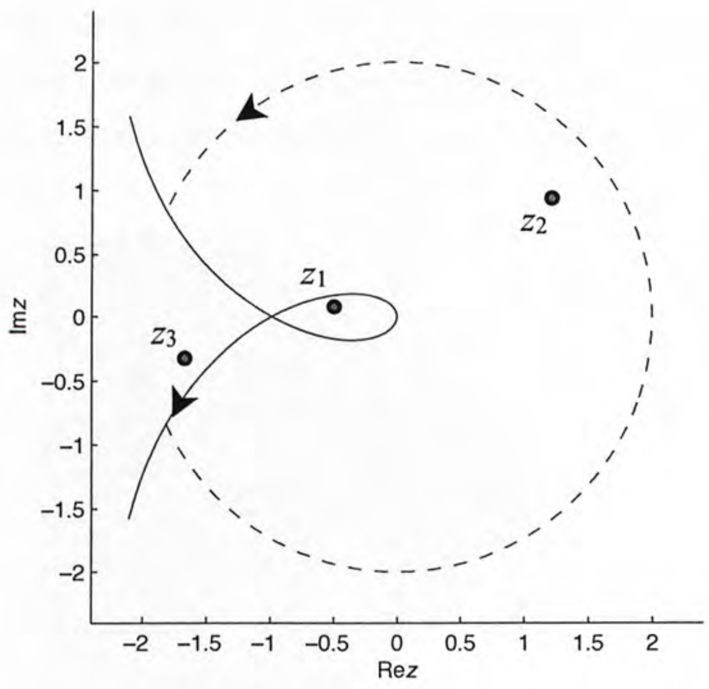

Figure 2.1: For example, let the symbol curve be given by the solid curve. By completing the symbol curve (for large enough $\overline{\omega}$) with a semicircle, denoted by the dashed curve, we find the winding number for the complex number z_1, z_2 and z_3 to be 2, 1 and 0, respectively.

by a semicircle centered at the origin, and then we have the winding number $I(\Phi_{\mathcal{A}}([-\overline{\omega}, \overline{\omega}]); z)$ for this closed curve. The winding number $I(\Phi_{\mathcal{A}}(\mathbb{R}); z)$ is taken to be the limit of $I(\Phi_{\mathcal{A}}([-\overline{\omega}, \overline{\omega}]); z)$ for large enough $\overline{\omega}$. See Figure 2.1 for illustration.

With the symbol curve and the winding number, we can now define the *critical region* of the jump-diffusion operator $\mathcal{A}$ by

$$\Sigma_{\mathcal{A}} = \{z \in \mathbb{C} : I(\Phi_{\mathcal{A}}(\mathbb{R}); z) \neq 1\}.$$

For example, in Figure 2.2 the critical region $\Sigma_{\mathcal{A}}$ is given by the shaded area. The critical region $\Sigma_{\mathcal{A}}$ is significant, as it characterizes the region where the *resolvent norm* $\|(z\mathcal{I} - \mathcal{A})^{-1}\|$ is large and the region where the resolvent norm is small (the threshold is roughly 1), cf. [43, 44, 56]. This is also illustrated by Figure 2.2. Therefore, in order for the validity of Dunford-Cauchy integral (2.16), a *feasible* Talbot contour Γ should enclose the critical region $\Sigma_{\mathcal{A}}$, i.e. $\Gamma \subset \mathbb{C} \backslash \text{closure}(\Sigma_{\mathcal{A}})$.

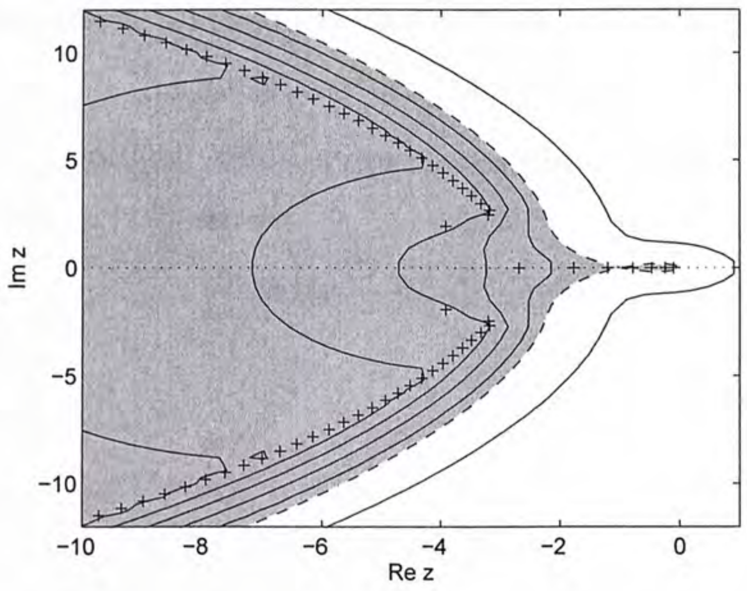

Figure 2.2: We plot the rightmost eigenvalues of $\mathbf{A}$ by the pluses. The contour plot of the resolvent norm $\|(z\mathcal{I}-\mathcal{A})^{-1}\|$ are marked by the solid lines. The contour levels, from left to right, represent $10^{10}, 10^{8}, ..., 10^{0}$. The symbol curve $\Phi_{\mathcal{A}}(\omega)$ is marked by the dashed line, and the associated critical region $\Sigma_{\mathcal{A}}$ is marked by the shaded area.

However, it remains unclear how the above feasibility constraint can be explicitly imposed on the parameters of parabolic Talbot contours. To answer this question, we need Theorem 2.4.2 in the following. Let B_{ε} denote the unit ball of radius ε centered at the origin.

Theorem 2.4.2. *Suppose* $0 < \delta < a_2$. *Then there exists a parabola*

$$\Psi(\omega) = -(a_2 - \delta)\omega^2 + ia_1\omega + a_0 + \rho, \ \omega \in \mathbb{R}, \tag{2.26}$$

with $\rho \geq \lambda$, so that its critical region $\Sigma_\Psi = \{z \in \mathbb{C} : I(\Psi(\mathbb{R}); z) \neq 1\}$ satisfies

$$\Sigma_{\mathcal{A}} + B_\varepsilon \subset \Sigma_\Psi,$$

for some $\varepsilon > 0$.

Proof. Denote by $\Phi_{\mathcal{D}}(\omega)$ the symbol curve of $\mathcal{D}$, i.e.

$$\Phi_{\mathcal{D}}(\omega) = -a_2\omega^2 + ia_1\omega + a_0, \ \omega \in \mathbb{R}.$$

Let $0 < \delta < a_2$ be given. We claim that there exists $\rho \geq \lambda$ so that the parabola $\Psi(\omega)$ of the form (2.26) satisfies

$$\min_{\omega_1,\omega_2} |\Phi_{\mathcal{D}}(\omega_1) - \Psi(\omega_2)| \geq \lambda + \varepsilon, \tag{2.27}$$

for some $\varepsilon > 0$.

We have

$$\begin{aligned}
& \min_{\omega_1,\omega_2} |\Phi_{\mathcal{D}}(\omega_1) - \Psi(\omega_2)|^2 \\
= \ & \min_{\omega_1,\omega_2} \left\{a_1^2(\omega_1-\omega_2)^2 + [a_2^2(\omega_1^2-\omega_2^2) + \delta\omega_2^2 + \rho]^2\right\} \\
\geq \ & \min_{\alpha,\beta,\omega_2} \left\{a_1^2\alpha^2 + (a_2^2\alpha\beta + \delta\omega_2^2 + \rho)^2\right\} \qquad (\alpha \equiv \omega_1 - \omega_2, \ \beta \equiv \omega_1 + \omega_2) \\
= \ & \min_{\alpha,\beta,\omega_2} \left\{(a_2^4\beta^2 + a_1^2)\left[\alpha + \frac{a_2^2\beta(\rho+\delta\omega_2^2)}{a_2^4\beta^2 + a_1^2}\right]^2 - \frac{a_2^4\beta^2(\rho+\delta\omega_2^2)^2}{a_2^4\beta^2+a_1^2} + (\rho+\delta\omega_2^2)^2\right\} \\
\geq \ & \min_{\beta,\omega_2} \left\{\frac{a_1^2(\rho+\delta\omega_2^2)^2}{a_2^4\beta^2+a_1^2}\right\}. \qquad \left(\text{take } \alpha = -\frac{a_2^2\beta(\rho+\delta\omega_2^2)}{a_2^4\beta^2+a_1^2}\right)
\end{aligned}$$

Without losing generality, assume $0 < \omega_1 \leq \omega_2$. Let $L > 0$ be fixed, then

(i) If $\omega_2 < L$,

$$\frac{a_1^2(\rho+\delta\omega_2^2)^2}{a_2^4\beta^2+a_1^2} \geq \frac{a_1^2\rho^2}{4L^2a_2^4 + a_1^2};$$

(ii) If $\omega_2 \geq L$,

$$\begin{aligned}
\frac{a_1^2(\rho+\delta\omega_2^2)^2}{a_2^4\beta^2+a_1^2} &\geq \frac{a_1^2(\rho+\delta\omega_2^2)^2}{4a_2^4\omega_2^2 + a_1^2L^{-2}\omega_2^2} \\
&= \frac{(\rho\omega_2^{-1}+\delta\omega_2)^2}{4a_2^4 + a_1^2L^{-2}} \\
&\geq \frac{2\rho\delta}{4a_2^4 + a_1^2L^{-2}}.
\end{aligned}$$

Combining (i) and (ii), we have

$$\min_{\omega_1,\omega_2} |\Phi_{\mathcal{D}}(\omega_1) - \Psi(\omega_2)|^2 \geq \min\left\{\frac{a_1^2\rho^2}{4L^2a_2^4 + a_1^2}, \frac{2\rho\delta}{4a_2^4 + a_1^2L^{-2}}\right\}.$$

Hence we can choose large enough ρ so that (2.27) holds. The claim is proved.

Moreover, since

$$\max_{\omega} |\Phi_{\mathcal{A}}(\omega) - \Phi_{\mathcal{D}}(\omega)| = \max_{\omega} \lambda \left|\int_{\mathbb{R}} f(y)e^{i\omega y}dy\right| \leq \lambda \int_{\mathbb{R}} |f(y)|dy = \lambda,$$

we have

$$\Sigma_{\mathcal{A}} \subset \Sigma_{\mathcal{D}} + B_{\lambda}.$$

Together with the claim, we conclude that

$$\Sigma_{\mathcal{A}} + B_{\varepsilon} \subset \Sigma_{\mathcal{D}} + B_{\lambda+\varepsilon} \subset \Sigma_{\Psi}.$$

□

In the following, we shall require a feasible Talbot contour Γ to enclose the critical region Σ_{Ψ} associated with the parabola $\Psi(\omega)$ in Theorem 2.4.2. This is sufficient to ensure that the Talbot contour encloses the original critical region $\Sigma_{\mathcal{A}}$. The effectiveness of this nonnormality control is illustrated by Figure 2.3 in the numerical experiments. Now we proceed onto how to parameterize a near-optimal parabolic Talbot contour among all feasible ones.

2.4.3 Optimal parameterization of parabolic Talbot contour

In this subsection, we devise optimal parameterization of parabolic Talbot contour. As in [57], this relies on the error estimate of the Talbot quadrature (2.19) in evaluating $(\varphi_l(\mathbf{A})\mathbf{v})_{l\geq 0}$. To ease our discussion, we introduce the following notations for (2.17)–(2.19)

$$G = \int_{-\infty}^{\infty} g(\theta)d\theta, \quad G_h = h\sum_{j=-\infty}^{\infty} g(\theta_j), \quad G_{h,N} = h\sum_{j=-N}^{N-1} g(\theta_j).$$

Let E^D and E^T denote the discretization error and the truncation error

$$E^D = \|G - G_h\|, \quad E^T = \|G_h - G_{h,N}\|. \tag{2.28}$$

The estimation on discretization error is given in the following theorem.

Theorem 2.4.3. *[57, Theorem 2.1] Let $w = \theta + ic$, with $\theta, c \in \mathbb{R}$. Suppose $g(w)$ is analytic in the strip $-d_- < c < d_+$, for some $d_+ > 0$, $d_- > 0$, and $g(w) \to 0$ uniformly as $|w| \to \infty$ in that strip. Suppose further that for some $M_+ > 0$, $M_- > 0$, the function $g(w)$ satisfies*

$$\int_{-\infty}^{\infty} \|g(\theta + ic_+)\| d\theta \le M_+, \qquad \int_{-\infty}^{\infty} \|g(\theta + ic_-)\| d\theta \le M_-,$$

for all $0 < c_+ < d_+$ and $-d_- < c_- < 0$. Then

$$\|G - G_h\| \le E_+^D + E_-^D,$$

where

$$E_+^D = \frac{M_+}{e^{2\pi d_+/h} - 1}, \qquad E_-^D = \frac{M_-}{e^{2\pi d_-/h} - 1}.$$

Finally, the truncation error E^T can be estimated by the magnitude of the integrand evaluated at the boundary of the truncation domain [57], i.e.

$$E^T = O\left(\|g(hN)\|\right), \quad \text{as } N \to \infty. \tag{2.29}$$

Now we are ready to parametrize the Talbot contour Γ as a parabola. Consider the holomorphic mapping

$$z = \mu(iw + 1)^2 + \gamma, \quad \text{where } \mu > 0. \tag{2.30}$$

The image of the horizontal line $w = \theta + ic, -\infty < \theta < \infty$, under (2.30), is

$$z = \mu[(1 - c)^2 - \theta^2] + \gamma + 2i\mu\theta(1 - c). \tag{2.31}$$

The contour Γ is given by the parabola (2.31) with $c = 0$. Weideman and Trefethen [57] considered the parameterization (2.30) with $\gamma = 0$ and allows $d_+ \to 1$.

Here we propose to add a horizontal shift γ in (2.30) for the following two reasons. First, the shift γ allows the parabola to satisfy the feasibility constraint that Γ encloses the critical region $\Sigma_{\mathcal{A}}$, see Theorem 2.4.2. Secondly, the shift γ allows $|z^{-l}| = O(1)$ so that the common-pole approximation to $(\varphi_l)_{l\geq 0}$ is valid.

To apply Theorem 2.4.3, let us consider (2.31) with $0 < c < d_+$. As d_+ increases from 0 to 1, the parabola closes and degenerates into the negative real axis. In the limiting case, a feasible parabola should enclose the critical region Σ_Ψ, see Theorem 2.4.2. This implies two constraints: one for the opening of the parabola μ and the other for the horizontal shift γ. The constraints are given explicitly as

$$\frac{4\mu(1-d_+)^2(a_2-\delta)}{a_1^2} \geq 1, \tag{2.32}$$

$$\mu(1-d_+)^2 + \gamma \geq a_0 + \rho, \tag{2.33}$$

for some valid ρ in Theorem 2.4.2. On the other hand, the condition $|z^{-l}| = O(1)$ for the common-pole approximation, see (2.21), implies that ρ should be bounded away from the origin. Note that as ρ increases from 0, we will sacrifice a fraction of accuracy in approximation of φ_0, but we will be compensated significantly with the accuracy in approximation of $(\varphi_l)_{l\geq 1}$. In our numerical experiment in Section 2.5, we will see that an appropriate ρ is available to strike a balance. In practice, we can choose $\rho = O(1)$ as suggested by Schmelzer [47].

According to Theorem 2.4.3, we have $E_+^D = O(e^{-2\pi d_+/h})$. Since we aim to minimize E_+^D (or to maximize d_+), we push $\delta \to 0^+$ and take the equality for (2.32), i.e.

$$d_+ = 1 - \frac{a_1}{2\sqrt{\mu a_2}}. \tag{2.34}$$

Thus, we have

$$E_+^D = O\left(\exp\left[-\frac{2\pi}{h}(1 - \frac{a_1}{2\sqrt{\mu a_2}})\right]\right), \quad \text{as } h \to 0.$$

Again consider (2.31) with $-d_- < c < 0$. As d_- increases from 0, the parabola widens and shifts towards right. Note that the growth of e^{Tz} contributes to the bound M_-, i.e. $M_-(d_-) = O(e^{T[\mu(1+d_-)^2+\gamma]})$. Then by Theorem 2.4.3, we have

$$E_-^D = O\left(\exp\left[T\mu(1+d_-)^2 + T\gamma - \frac{2\pi d_-}{h}\right]\right).$$

When $d_- = \pi/(T\mu h) - 1$, E_-^D attains the minimum

$$E_-^D = O\left(\exp\left[-\frac{\pi^2}{T\mu h^2} + T\gamma + \frac{2\pi}{h}\right]\right), \quad \text{as } h \to 0.$$

Finally, the truncation error E^T is estimated according to (2.29) as

$$E^T = O\left(\exp\left[T\mu(1-h^2N^2) + T\gamma\right]\right), \quad \text{as } h \to 0.$$

Since we are minimizing E_-^D and E^T (or minimizing γ), we take the equality for (2.33). Together with (2.34), we have

$$\gamma = a_0 + \rho - \frac{a_1^2}{4a_2}.$$

As in [57], we treat the min-max problem

$$\min_{\mu,h} \max\{E_+^D, E_-^D, E^T\},$$

approximately by balancing the asymptotical rates

$$-\frac{2\pi}{h}\left(1 - \frac{a_1}{2\sqrt{\mu a_2}}\right) = -\frac{\pi^2}{T\mu h^2} + T\gamma + \frac{2\pi}{h} = T\mu(1-h^2N^2) + T\gamma =: E(N). \quad (2.35)$$

Solving the scalar equations (2.35), e.g. by the Matlab routine `solve`, we obtain the optimal parameters, μ^* and h^*, for the parabolic Talbot contour.

Thus, given the parameterization of the Talbot contour, we can obtain the poles and the weights in (2.20), plug into the quadrature formula (2.24), and compute the numerical solution of the option price. We remark that the cost for computing the optimal parameters is negligible, compared with the main cost of solving N large linear systems in (2.24). However, the parameterization is crucial for accurate numerical solution, as we will see in the numerical experiments, see e.g. Figure 2.7.

2.5 Numerical experiments

Example 1

In the first example, we test the proposed parallel Talbot method (PTM) on a European call option under Merton's lognormal model [39], where the jump distribution $f(x)$ in (2.1) is given by

$$f_{\text{Merton}}(x) = \frac{1}{\sqrt{2\pi}\bar{\sigma}} \exp\left(-\frac{(x-\bar{\mu})^2}{2\bar{\sigma}^2}\right).$$

The following parameters are used: $K = 100$, $r = 0.05$, $q = 0.02$, $\sigma = 0.1$, $T = 1$, $\lambda = 2$, $\bar{\mu} = -0.3$, $\bar{\sigma} = 0.4$, and $\kappa = e^{\bar{\mu}+\bar{\sigma}^2/2} - 1$. We take the truncation boundary $x_{\min} = -3$, $x_{\max} = 3$ and the mesh size $\Delta x = 10^{-3}$ (or $M = 6{,}000$). We are interested in the option price at $S_0 = K$. Except for Figure 2.5, all the errors in the numerical approximation are calculated with the exact price 24.503308 (by analytical formula [39]).

In Figure 2.3, we verify our estimation for the critical region $\Sigma_{\mathcal{A}}$, see Theorem 2.4.2. Observe that the eigenvalues of $\mathbf{A}$ move off the negative real axis. We remark that in our experiment the condition $\rho \geq \lambda$ is sufficient and necessary for the parabola $\Psi(\omega)$ to enclose $\Sigma_{\mathcal{A}}$.

Figure 2.4 reveals the failure of the Talbot contour in [57], as it intersects the critical region $\Sigma_{\mathcal{A}}$ and traverses the spectrum of $\mathbf{A}$. Our remedy is to include a shift γ in (2.30) and impose the feasibility constraints, see (2.32)–(2.33). The devised Talbot contour is plotted by the solid line with 16 quadrature nodes marked with dots.

In Figure 2.5, we further the comparison of the contour in [57] and the contour we propose. We test the scalar approximations by the two contours to $\varphi_0(z)$ and $\varphi_1(z)$, see (2.19), on the vertical line $\operatorname{Re} z = -5$. Our observation is that the accuracy by the contour in [57] is higher when z lies on the negative real axis, but soon decreases as z moves away from the real axis. To the contrary, the accuracy

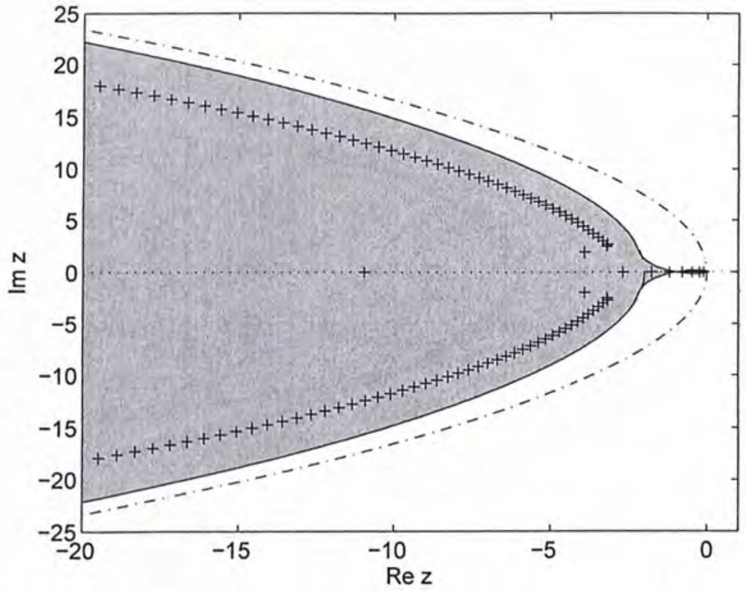

Figure 2.3: We plot the rightmost eigenvalues of $\mathbf{A}$ by the pluses, the symbol curve $\Phi_{\mathcal{A}}(\omega)$ by the solid line, its asymptotical approximation $\Phi_{\mathcal{D}}(\omega)$ by the dashed line (which coincides with the symbol curve asymptotically), the critical region $\Sigma_{\mathcal{A}}$ by the shaded area, and the parabola $\Psi(\omega)$ in Theorem 2.4.2, with $\rho = \lambda$ and $\delta = 0^+$, by the dash-dotted line.

by our contour is more uniform (as the curves are almost flat in the figure) inside the region Σ_{Ψ}, or $|c| \leq 13.84$ according to Figure 2.4. This observation provides a heuristic explanation on why our proposed contour is superior to that in [57] for the nonnormal problem.

Back to PTM for the option pricing problem, Figures 2.6–2.8 concern optimal parameter selection and error estimate (for option prices). Recall that the common-pole approximation for φ_0 and φ_1 require $\rho = O(1)$. As ρ increases from 0, we sacrifice the accuracy in approximation of φ_0, but gain the accuracy in approximation of φ_1. As ρ becomes even larger, the accuracy in both approximations will drop as expected. In Figure 2.6, we find that it strikes a good balance

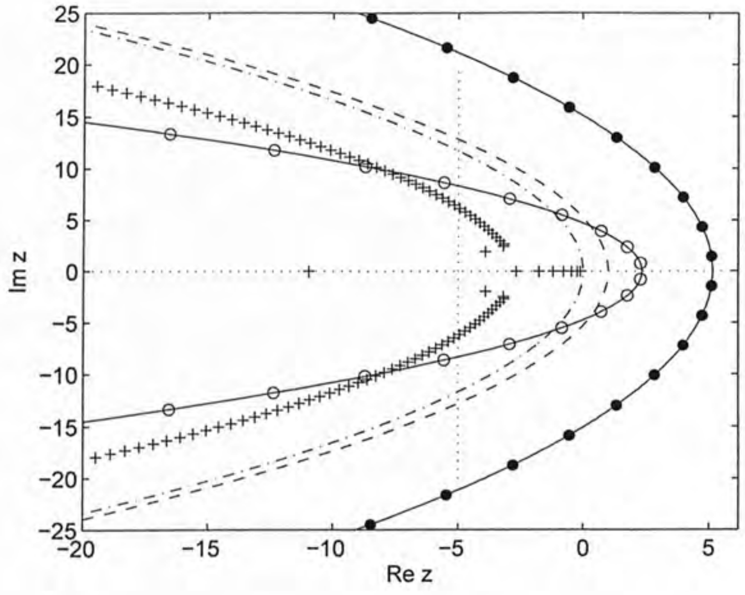

Figure 2.4: The pluses and the dash-dotted line are the same as in Figure 2.3. The parabolic contour proposed in [57] is plotted by the line marked with circle. The parabolic contour we propose for nonnormal problem is plotted by the line marked with dot. In the limiting case, the parabola (2.31) with $c = d_+$, see (2.34), is plotted by the dashed line. The dotted vertical line is Re $z = -5$.

to take $\rho = 2$ (marked by the cross) for Example 1.

Figure 2.7 accredits our strategy to estimate the optimal parameters μ^* and h^*. In the left part of the figure, or for small μ, the error E_+^D dominates; in the northeast sections, the error E_-^D dominates; and in the south and southeastern sections, E^T dominates. Although our estimate in Section 2.4.3 is asymptotical, the resulted parameters, $\mu^* = 10.2419$ and $h^* = 0.1430$, are indeed near-optimal for small $N(= 8)$.

Figure 2.8 illustrates the convergence of PTM. Note that we can calculate $E(N)$ by solving μ and h in (2.35) for each positive integer N. Then we have a priori error estimate given as $O(e^{-E'(N)N})$, where $E'(N) \approx E(N) - E(N-1) =$

-1.5014 in this example. We observe that our priori estimate is consistent with the experiment data.

Example 2

We address the issues of implementation and complexity with a second example, where we test PTM on an up-and-out put option under Kou's double exponential model [30]. The jump distribution $f(x)$ in (2.1) is given by

$$f_{\text{Kou}}(x) = \begin{cases} p\eta_1 e^{-\eta_1 x}, & \text{if } x \geq 0; \\ (1-p)\eta_2 e^{\eta_2 x}, & \text{if } x < 0. \end{cases}$$

As we will compare the performance of our method and that of the extrapolation-implicit-explicit method (ext-IMEX) in Feng and Linetsky [20], we use the parameter set therein: $K = 100$, $r = 0.05$, $q = 0.02$, $\sigma = 0.1$, $T = 1$, $\lambda = 3$, $p = 0.3$, $\eta_1 = 40$, $\eta_2 = 12$, the upper knock-out barrier $U = 120$, and $\kappa = p(\eta_1 - 1)^{-1} - (1-p)(\eta_2 + 1)^{-1}$. In this example, we set $x_{\min} = -2$, $x_{\max} = \log(U/K)$, $\rho = 1$. Again, we are interested in the option price at $S_0 = K$, for which the exact price is 5.75775 (by standard Crank-Nicolson method).

In PTM, we need to solve N independent linear systems, see (2.24). In our experiment, each linear system is solved by the generalized minimal residual method (GMRES) with left preconditioner $z_j\mathbf{I} - \mathbf{B}$. Given the spatial discretization in Section 2.3, we construct the tridiagonal matrix $\mathbf{B}$ by matching the tridiagonals of $\mathbf{A}$. The effectiveness of this simple preconditioning strategy is illustrated by Table 2.1. For a fixed residual tolerance 10^{-13}, the iteration numbers of preconditioned GMRES for $N(=8)$ linear systems are independent with the matrix size M. Since each matrix-vector multiplication is of complexity $O(M \log M)$ due to fast Fourier transform (FFT), and since the inversion of a tridiagonal system can be done with $O(M)$ operations by sparse matrix solvers, the total complexity of PTM is therefore $O(NM \log M)$.

Table 2.1: Iteration numbers of preconditioned GMRES

	the j-th linear system							
M	1	2	3	4	5	6	7	8
682	14	14	13	12	12	11	10	9
1364	14	13	13	12	11	11	10	9
2728	13	13	12	12	11	10	10	9
5456	13	12	12	11	11	10	9	9
10912	12	12	11	11	10	9	9	8
21824	12	11	11	10	10	9	8	8

In the following, we compare the performance of PTM and that of ext-IMEX [20]. For both methods, we assume the same spatial discretization with fixed mesh size $\Delta x = 10^{-4}$, or $M = 10912$, yielding a minimal error around 2×10^{-6}. In the s-level ext-IMEX, we use the 1st-order implicit-explicit Euler scheme with initially 10 time steps. For the j-th level, $j = 1, ..., s$, we do the time marching with $10j$ times steps. The last step is to combine the results at all s levels and complete the extrapolation tableau. Note the total number of time steps in ext-IMEX $n \approx s^2/2$ and the error $\mathcal{E}_{\text{ext-IMEX}} = O(e^{-C_1\sqrt{n}\log n})$, cf. [20]. As the runtime $\mathcal{T}_{\text{ext-IMEX}}$ is proportional to n, we have the following estimate for ext-IMEX

$$\mathcal{E}_{\text{ext-IMEX}} = O(\exp[-C_2(\mathcal{T}_{\text{ext-IMEX}})^{1/2}\log(\mathcal{T}_{\text{ext-IMEX}})]). \tag{2.36}$$

In PTM, we balance the approximation error of Talbot quadrature and that of GMRES iteration. Our choice is to take the number of preconditioned GMRES iterations m, to be proportional to $N + \beta$, where N is the number of quadrature nodes and β is some positive integer. Note that the total amount of work is $O(mN)$, or $O(N^2 + \beta N)$. Therefore, the complexity of $\mathcal{T}_{\text{Talbot}}(N)$ is at almost quadratic. Nevertheless, it can also be estimated as a power function $\mathcal{T}_{\text{Talbot}} = O(N^\alpha)$, where $1 < \alpha < 2$. Together with the exponential convergence of Talbot quadrature, we have the following estimate for PTM

$$\mathcal{E}_{\text{Talbot}} = O(\exp[-C_3(\mathcal{T}_{\text{Talbot}})^{1/\alpha}]).$$

Figure 2.9(a) shows the experiment results on a single-processor machine. In this experiment, we find approximately $\alpha = 1.2838$, or $1/\alpha = 0.7790 > 0.5$. Although PTM is more favorable in terms of the theoretical estimate, the figure shows that ext-IMEX outperforms PTM before both two are close to attain the minimum error around 10^{-6}.

Now we consider the parallel implementations with minimum communication for both ext-IMEX and PTM. We assume that both methods attain reasonably good accuracy with s (for ext-IMEX) and N (for PTM) less than the maximum number of processors available. For ext-IMEX, we distribute totally s independent levels of time marching schemes to s nodes. After all time marching schemes finish, we collect the results from each computing node and complete the extrapolation tableau. Note that the error estimate $\mathcal{E}_{\text{ext-IMEX}} = O(e^{-C_1\sqrt{n}\log n})$ remains valid, but the runtime $\mathcal{T}_{\text{ext-IMEX}}$ becomes proportional to s, since it is determined by the level with the most time steps. With $n = s^2/2$, the estimate (2.36) changes accordingly

$$\mathcal{E}_{\text{ext-IMEX}} = O(\exp[-C_4\mathcal{T}_{\text{ext-IMEX}}\log(\mathcal{T}_{\text{ext-IMEX}})]).$$

For PTM, we solve totally N independent linear systems by preconditioned GMRES iteration on N computing nodes separately, and then sum the results as in the formula (2.24). Since the work is distributed evenly to N nodes, the runtime $\mathcal{T}_{\text{Talbot}}$ reduces to a multiple of $N + \beta$. So

$$\mathcal{E}_{\text{Talbot}} = O(\exp[-C_5\mathcal{T}_{\text{Talbot}} + C_6\beta]).$$

In Figure 2.9(b), we observe that for the error less than around 10^{-4}, PTM outperforms ext-IMEX. In fact, the slope (in magnitude) of the log-error for PTM nearly doubles that for ext-IMEX. Due to the effect of the intercept β, ext-IMEX is more suitable for low-accuracy computation.

The experiments are implemented on the Linux Cluster—Organon [1], with each computing node installed with Matlab v7.1.0.183 (R14) Service Pack 3 and MatlabMPI [2].

2.6 Conclusion

In this chapter, we introduce a novel parallel Talbot method (PTM) for solving the initial-boundary value problem arising from the jump-diffusion model in option pricing. PTM yields a geometrically convergent time quadrature, and therefore outperform traditional time-marching schemes of polynomial-order convergence. In particular, we devise optimal parameterization of parabolic Talbot contour, by balancing the asymptotical error decay with feasibility constraints. Our contributions are two-fold: first, we derive explicit conditions on the control of the (pseudo)spectrum of nonnormal jump-diffusion operators; second, we consider common-pole approximation for general φ-functions, thus greatly saving computational cost. In the numerical experiment, we see that the optimal parameterization of Talbot contour is effective,and that PTM is competitive on parallel machine.

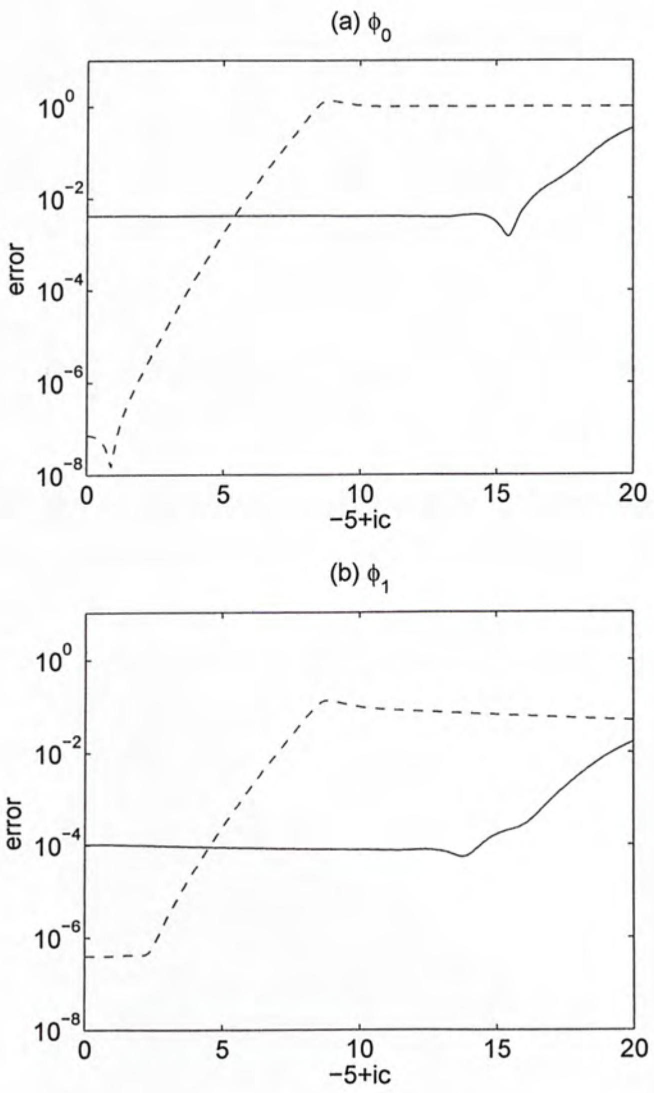

Figure 2.5: We plot the errors of scalar problems (rather than option pricing) for approximating $\varphi_0(-5+ic)$ in (a) and $\varphi_1(-5+ic)$ in (b), where $0 \leq c \leq 20$ for both. In each figure, the dashed line is the error for the contour in [57] and the solid line is that for our proposed contour.

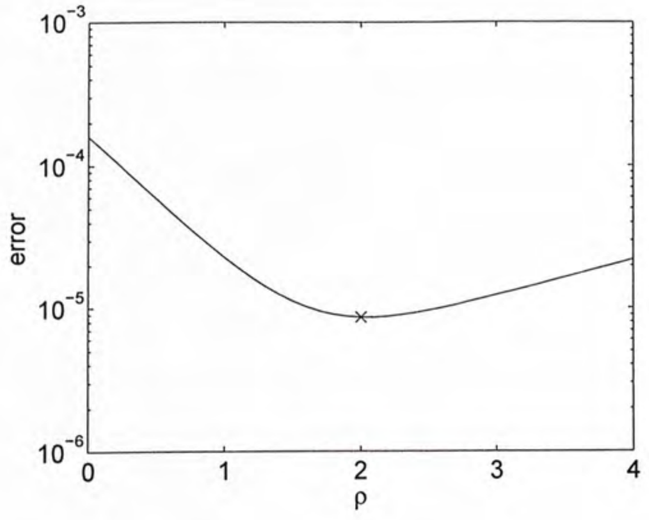

Figure 2.6: We plot the error as the shift parameter ρ varies. The optimal choice $\rho = 2$ is marked by the cross.

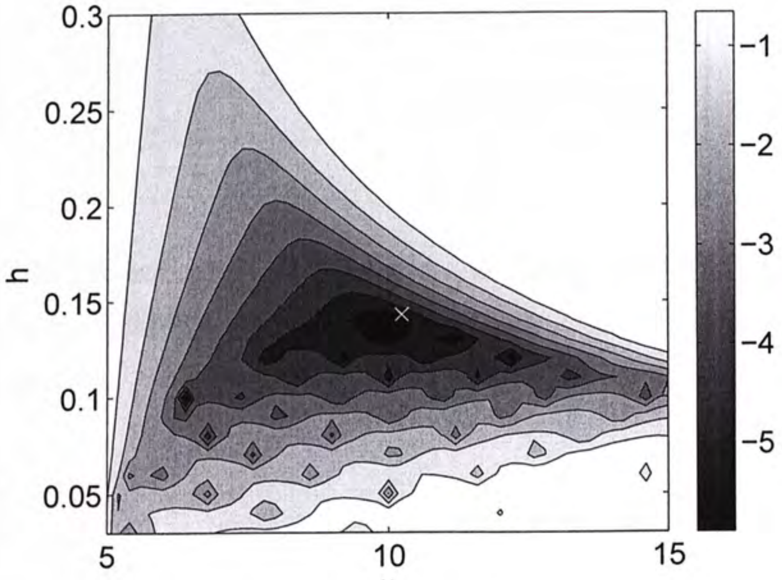

Figure 2.7: This is the contour plot of $\log_{10} |\mathcal{E}_{\text{Talbot}}|$, with $N = 8,\ \rho = 2$, for different choices of μ and h. Our parameter estimate that $\mu^* = 10.2419,\ h^* = 0.1430$, see (2.35), is marked by the cross.

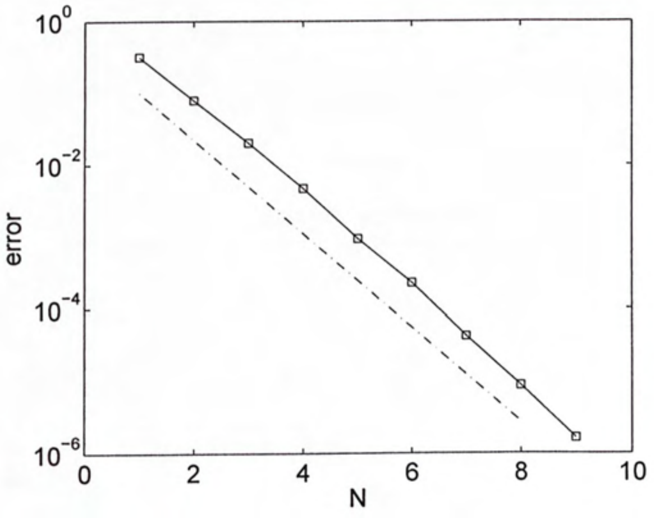

Figure 2.8: The error, with optimal choice of parameters, $\mu^*(N)$ and $h^*(N)$, is plotted by the solid line. The priori error estimate $O(e^{-1.5014N})$ is plotted by the dash-dotted line.

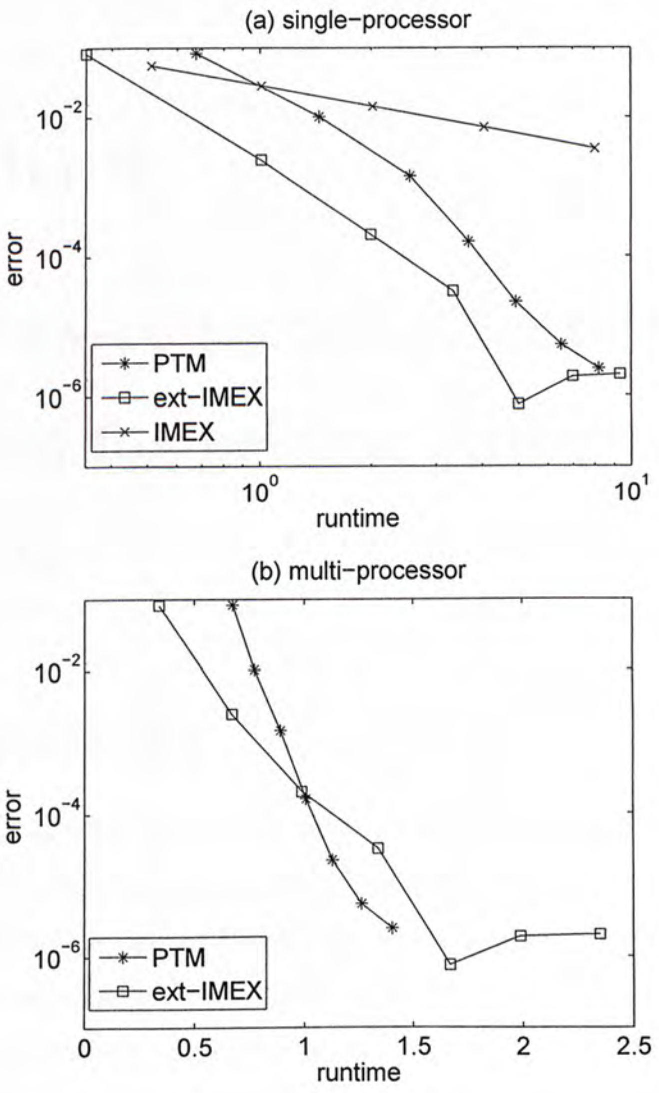

Figure 2.9: We plot the runtime (in second) vs the error for the parallel Talbot method ($N = 1, ..., 7$), those for ext-IMEX ($s = 1, ..., 7$), and those for the 1st-order IMEX method (only in (a)). The experiment on a single-processor machine is shown in (a) and that on a multi-processor machine (with a maximum of eight processors) is shown in (b).

Chapter 3

Memory-reduction Monte Carlo method for pricing American options

3.1 Introduction

It is well known, see e.g. [24], that with the arbitrage-free principle the option price is given by the discounted expected payoff under certain risk-neutral measure. This leads to option pricing by the Monte Carlo method, for which the first application was made by Boyle [9] in 1977. Since then, Monte Carlo method has been a popular tool in pricing financial derivatives [24]. Yet, Monte Carlo method is known to have difficulties in handling American-style options with early exercise feature. In 2001, Longstaff and Schwartz [32] proposed a practical algorithm, named least squares method (LSM), to price American options. Their method is based on a backward-in-time induction, where at each time step the continuation value of the option is estimated by least squares regression.

However, one drawback of LSM is that, in order to compute the intermediate exercise prices at all time steps, it requires the storage of all asset prices at all time steps for all simulated paths. Thus the total storage requirement grows like $O(mn)$ where m is the number of time steps and n is the number of simulated paths. The plain Monte Carlo method, referred as the *full-storage method* in this chapter, is therefore computationally inefficient since the accuracy of the simulation is severely limited by the storage requirement.

This storage problem can be alleviated by "bridge methods" such as the Brownian bridge [12], the inverse Gaussian bridge [45], and the gamma bridge [46] — where the memory requirement can be reduced to $O(n \log m)$. Nevertheless, one drawback is that a specific bridge method can only work on the corresponding model that the price of the underlying asset follows. Thus the Brownian bridge is suitable for Brownian motion, the gamma bridge for the variance gamma process, and so on. That is to say, all bridge methods are model-dependent, which limits their use in applications.

In this chapter, we develop a memory-reduction method, which does not require storing of all intermediate asset prices. The storage is significantly reduced to $O(m + n)$. Coupled with the least squares method proposed in [32], our memory-reduction method is applicable to the general class of exponential Lévy processes. The main idea of our method is to first generate the price process forward until the expiration time, and to store only the *seeds* of the random number sequences at each time step. When computing the option prices backwardly, we recompute the just-in-time asset prices using the corresponding seeds. Since the prices are recomputed exactly, the memory-reduction method gives the same result as the full-memory method. The additional computational cost is the cost of regenerating the random numbers corresponding to the asset prices. The total computational cost is therefore always less than twice that of the full-storage method.

The remainder of the chapter is organized as follows. Section 2 reviews the exponential Lévy processes as well as the full-storage method. Section 3 gives the background of random number generators and the concept of seeds. Section 4 introduces our memory-reduction method. In Section 5, we show how the memory-reduction method is applied to specific models — viz. the Black-Scholes model, Merton's jump-diffusion model and the variance gamma model. Numerical results are provided there to show the efficiency and accuracy of our method, by comparing it with methods from other well-known approaches. Concluding remarks are drawn in Section 6.

3.2 Exponential Lévy processes and the full-storage method

Let the risk-neutral price dynamics be modeled by the exponential Lévy process

$$S_t = S_0 \exp(rt + L_t), \tag{3.1}$$

with the risk-free rate r and a Lévy process L_t. A Lévy process L_t is a stochastic process with stationary independent increments, continuous in probability, having sample paths that are right-continuous with left limits ("cadlag"), and satisfying $L_0 = 0$. We note that the increments, $L_s - L_t$ for any $s > t$, are independent if the increments $L_s - L_t$ and $L_u - L_v$ are independent random variables whenever the two time intervals $[t, s]$ and $[v, u]$ do not overlap. The increments are stationary if the distribution of any increment $L_s - L_t$ only depends on $s - t$; and therefore increments with equally long time intervals are identically distributed.

We first review the Monte Carlo simulation for computing American-style options. First the time horizon is discretized into m time steps with equal length $\Delta t := (T - t_0)/m$ as $t_0 < t_1 < \ldots < t_m = T$, or $t_j = t_0 + j\Delta t$, where t_0 is the current time and T is the expiration date of the option. Let $L_{i,j}$ denote the realization of L_t on the i-th path at time t_j. They are computed by adding the increment $\Delta L_{i,j} := L_{i,j} - L_{i,j-1}$ to $L_{i,j-1}$ recursively at each time step. Thus the whole path simulation process is to simulate the random numbers that give $\Delta L_{i,j}$. We will denote by $\Sigma_{i,j} = \{\varepsilon_{i,j}^k\}_{k=1}^{\eta_{i,j}}$ the ordered set of [0,1] uniform random numbers used in generating $\Delta L_{i,j}$. Here $\eta_{i,j}$ is the number of random numbers required to generate $\Delta L_{i,j}$. It is different for different process. The outline for a general of path simulation procedure is given below:

Algorithm 3.2.1. *(Path simulation)*

```
For-loop: $i = 1, 2, ..., n$
    Set $L_{i,0} \leftarrow 0$
    For-loop: $j = 1, 2, ..., m$
        1. Get the increment $\Delta L_{i,j}$ by generating $\Sigma_{i,j}$
        2. $L_{i,j} \leftarrow L_{i,j-1} + \Delta L_{i,j}$
        3. $S_{i,j} \leftarrow S_0 \exp(rj\Delta t + L_{i,j})$
    End for-loop
End for-loop
```

Algorithm 3.2.1 simulates the paths and then stores all intermediate asset prices $S_{i,j}$ for later computation of the option prices, hence the storage requirement grows like $O(mn)$. We call this the full-storage method. Once we have all the intermediate asset prices $S_{i,j}$, we can price American-style options using the least square method (LSM) suggested by [32]. Let us recall it here. At the final exercise date T, the optimal exercise strategy for an American option is to exercise it if it is in the money. This can be done as the terminal asset prices $S_{i,m}$ are available for each path i. However, prior to T the optimal strategy is to compare the immediate exercise value with the expected cash flows from continuing, and then exercise if immediate exercise is more valuable. In the full-storage method, the intermediate asset prices $S_{i,j}$ are available for each path i and at each time step j. Thus the key to optimally exercising an American option is to identify the conditional expected value of continuation. In [32], the cross-sectional information in the simulated paths is used to identify the conditional expectation function. This is done by regressing the cash flows from continuation on a set of basis functions depending on the current asset prices $S_{i,j}$. The fitted function from this regression is an efficient unbiased estimate of the conditional expectation functions, from which one can estimate an optimal stopping rule for the option.

Numerical illustration of LSM for pricing American put options under the Black-Scholes framework can be found for instance in [32]. The computational complexity of the full-storage method is $O(mn)$.

3.3 Random number generators

In Step 1 of Algorithm 3.2.1, in order to get $\Delta L_{i,j}$ we need to generate a set of [0,1] uniform random numbers $\{\Sigma_{i,j}\}$ for each time step j on each path i. Most programming softwares already have built-in functions to generate $[0,1]$ uniform random numbers. In MATLAB, we can initialize the pseudorandom number generator with seed `d` by the command `rand('seed',d)`, and then generate a pseudorandom sequence $\{\varepsilon_k\}$ by repeatedly using the command `rand`. In MATLAB, $\{\varepsilon_k\}$ is generated by a simple multiplicative congruential generator [41, Chapter 9]

$$d_0 = \mathtt{d}, \quad d_k = a d_{k-1} + c \bmod M, \text{ for } k \geq 1; \quad \varepsilon_k \equiv d_k / M. \tag{3.2}$$

The parameters in (3.2) are chosen as $a = 16807$, $c = 0$, $M = 2^{31} - 1$, due to Park and Miller [42].

Thus a pseudorandom sequence is actually not random but deterministic, in the sense that it is generated according to some formula and hence can be regenerated exactly if the seed d_0 is known. For example, the MATLAB commands

```
rand('seed',d);
e=rand;
```

will output different e if the seed d is changing every time, but output the same e if d is fixed. By extracting and remembering a proper seed, we can regenerate part of a pseudorandom sequence as we desire. More specifically, suppose we have already generated a sequence $\{\varepsilon_k\}_{k=1}^{p}$, and then we want to regenerate only $\{\varepsilon_k\}_{k=q}^{p}$, i.e. the part of the sequence beginning at ε_q. All we need is to extract the seed after generating ε_{q-1}. The seed-extracting command in MATLAB is `rand('seed')`. Thus given the sequence $\{\varepsilon_k\}_{k=1}^{p}$ generated by

$$\xrightarrow{\texttt{randn}} \varepsilon_1 \ldots \xrightarrow{\texttt{randn}} \varepsilon_{q-1} \xrightarrow{\texttt{c=randn('seed')}} \text{extract seed } \texttt{c} \xrightarrow{\texttt{randn}} \varepsilon_q \ldots \xrightarrow{\texttt{randn}} \varepsilon_p,$$

we can regenerate $\{\varepsilon_k\}_{k=p}^{q}$ by

$$\xrightarrow{\texttt{randn('seed',c)}} \text{set seed } \texttt{c} \xrightarrow{\texttt{randn}} \varepsilon_q \xrightarrow{\texttt{randn}} \varepsilon_{q+1} \xrightarrow{\texttt{randn}} \ldots\ldots \xrightarrow{\texttt{randn}} \varepsilon_p$$

Some computer languages only provide [0,1] uniform random numbers. When we simulate Lévy processes, we will also need to generate non-uniform random variables such as the standard normal random variables, Poisson random variables, and the gamma random variables. Various kinds of methods, say the inverse transform method and the acceptance-rejection method, can be used to obtain non-uniform random variables based on [0,1] uniform random numbers. For standard normal random numbers, the most commonly used method is the Box-Muller transformation [17, pp. 235]. For Poisson random variables, the inverse transform method is a standard method [24, pp. 128]. For completeness, we provide the Best's generator for the gamma random variables in the Appendix, cf. [17, pp. 410 and pp. 420]. We will be using these methods to generate the needed random variables. In the following, we will use $Z \sim \mathcal{N}(0,1)$ and $\varepsilon \sim \mathcal{U}[0,1]$ to denote random numbers Z and ε distributed as standard normal and [0,1] uniform respectively.

3.4 The memory-reduction method

In this section, we present our memory-reduction method which does not require one to store the intermediate asset prices $\{S_{i,j}\}_{i,j=1}^{n,m}$ when computing the option prices. In this method, each increment $\Delta L_{i,j}$ is generated twice without being stored while the corresponding intermediate asset price $S_{i,j}$ is generated only once in the backward pricing of the option.

As in the full-storage method, we compute $L_{i,j} = L_{i,j-1} + \Delta L_{i,j}$ by using the increments $\Delta L_{i,j}$. But in our memory-reduction method, we use a different way to generate the set of random numbers $\Sigma_{i,j}$ to obtain $\Delta L_{i,j}$—we generate them time-wise. More precisely, we obtain the increments $\Delta L_{i,1}$ by generating the random numbers in $\Sigma_{i,1}$ on each path i, $i = 1, ..., n$, for the time step $j = 1$ first. Then we obtain $\Delta L_{i,2}$ by generating $\Sigma_{i,2}$ on all paths for $j = 2$, etc. For each time step j, at the last path, i.e. path n, we extract and save the current seed d_j for later use. Given an arbitrary seed d_1, the procedures can be illustrated as follows (cf. Phase 2 in the following Algorithm 3.4.1):

$$\begin{array}{l}
\text{set seed } d_1 \to \Delta L_{1,1}(\Sigma_{1,1}) \to \Delta L_{2,1}(\Sigma_{2,1}) \to \dots\dots \to \Delta L_{n,1}(\Sigma_{n,1}) \to \\
\text{extract seed } d_2 \to \Delta L_{1,2}(\Sigma_{1,2}) \to \Delta L_{2,2}(\Sigma_{2,2}) \to \dots\dots \to \Delta L_{n,2}(\Sigma_{n,2}) \to \\
\text{extract seed } d_3 \to \dots\dots \\
\vdots \\
\text{extract seed } d_m \to \Delta L_{1,m}(\Sigma_{1,m}) \to \Delta L_{2,m}(\Sigma_{2,m}) \to \dots\dots \to \Delta L_{n,m}(\Sigma_{n,m})
\end{array}$$

Note that we need an m-vector to hold $\{d_j\}_{j=1}^{m}$ and an n-vector to hold $\{L_{i,j}\}_{i=1}^{n}$. That n-vector can be re-used for every time step j.

When computing the option price we move backward in time, and compute on each path i the corresponding asset prices $S_{i,j} = S_0 \exp(rj\Delta t + L_{i,j})$ at each time step j. This requires $L_{i,j}$. Given $L_{i,j+1}$, to obtain $L_{i,j}$, we only need to regenerate $\Delta L_{i,j+1}$. This can be done by reproducing the random number sequence in $\Sigma_{i,j+1}$ using the seed d_{j+1}, i.e.

$$\text{set seed } d_j \rightarrow \Delta L_{1,j+1}(\Sigma_{1,j+1}) \rightarrow \ldots\ldots \rightarrow \Delta L_{n,j+1}(\Sigma_{n,j+1})$$

Once we get all the $S_{i,j}$ for the time step j, we can compute the option prices on all paths at time step j by using the LSM method in [32]. We summarize our memory-reduction method in Algorithm 3.4.1 below:

Algorithm 3.4.1.

```
Phase 1 (path simulation):
Set $L_0^i \leftarrow 0$ for $i = 1, 2, ..., n$
For-loop: $j = 1, 2, ..., m$
    1. Extract the current seed $d_j$
    For-loop: $i = 1, 2, ..., n$
        2. Get the increment $\Delta L_{i,j}$ by generating $\Sigma_{i,j}$
        3. $L_{i,j} \leftarrow L_{i,j-1} + \Delta L_{i,j}$
    End for-loop
End for-loop

Phase 2 (price computation):
For-loop: $j = m, ..., 1$
    If $j < m$,
        4. Recall the seed $d_{j+1}$
        For-loop: $i = 1, 2, ..., n$
            5. Get the increment $\Delta L_{i,j+1}$ by regenerating $\Sigma_{i,j+1}$
            6. $L_{i,j} \leftarrow L_{i,j+1} - \Delta L_{i,j+1}$
            7. $S_{i,j} \leftarrow S_0 \exp(rj\Delta t + L_{i,j})$
        End for-loop
    End if
    Compute the current option price on all paths using the LSM method
End for-loop
```

We note that our memory-reduction approach requires only three vectors: an m-vector for storing the seeds $\{d_j\}_{j=1}^m$ in Steps 1 and 4, an n-vector to hold $\{L_{i,j}\}_{i=1}^n$ for the current time-step j in Steps 3 and 6 and an n-vector to hold $\{S_{i,j}\}_{i=1}^n$ for the current time-step j in Step 7. The additional computational burden is Steps 1–4 in Phase 1, where we generate the paths and remember the seeds. Since in Phase 2 we are regenerating the exact paths as in the full-storage method, it is clear that the results obtained by the full-storage method and the memory-reduction method are exactly the same. Moreover, since path generation is only one part of all the computations required in the algorithm (the other part—the major part—being the least-squares methods of [32]), we see that the total cost of our method is less than twice that of the full-storage method. We will illustrate these facts numerically in Section 3.5. We note that in order to use our Algorithm 3.4.1 for different kinds of option, we only need to specify how $\Delta L_{i,j}$ in Step 2 are generated.

3.5 Numerical examples

In this section, we apply our method to different models in the class of exponential Lévy processes. In Subsection 3.5.1, we consider the Black-Scholes model and compare our memory-reduction method with the Brownian-bridge method and also the Crank-Nicolson method. In Subsections 3.5.2 and 3.5.3, numerical results are reported for both finite-activity and infinite-activity jump processes, respectively. We compare our results with a binomial tree method and an integro-differential equation method. Regarding the LSM we used, we estimate the continuing values of an option on those "in-the-money" samples and choose the first three Laguerre polynomials plus a constant term as our basis functions throughout the section.

3.5.1 Black-Scholes model

As an illustration for how to use the memory-reduction method, we begin with the Black-Scholes model:

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t, \tag{3.3}$$

where r is the risk-free rate, σ is the volatility, and W_t is the standard Wiener Process. The memory-reduction method for this simple case was considered in [10, 11], but we repeat it here as an introduction to our method. By Itô's lemma, the L_t in (3.1) becomes $L_t = -\frac{1}{2}\sigma^2 t + \sigma W_t$ and hence

$$\Delta L_{i,j} = -\frac{1}{2}\sigma^2 \Delta t + \sigma\sqrt{\Delta t} Z_{i,j} \tag{3.4}$$

where $Z_{i,j} \sim \mathcal{N}[0,1]$. By the Box-Muller transformation [17, pp. 235], a pair of $Z_{i,j}$ can be generated by a pair of $\varepsilon_{i,j} \sim \mathcal{N}[0,1]$. Hence here the set $\Sigma_{i,j}$ in Algorithm 3.4.1 has only one element $\varepsilon_{i,j}$. Now we can apply Algorithm 3.4.1 by specifying the procedures in Step 2 as follows:

Algorithm 3.5.1 (Black-Scholes).

```
1. Generate $Z_{i,j} \sim \mathcal{N}(0,1)$ using $\varepsilon_{i,j} \sim \mathcal{U}[0,1]$
2. $\Delta L_{i,j} \leftarrow -\frac{1}{2}\sigma^2 \Delta t + \sigma\sqrt{\Delta t} Z_{i,j}$
```

Next we compare our memory-reduction method with the Brownian-bridge method in [12] and the Crank-Nicolson method on pricing American put options under model (3.3). Note that the results obtained by the full-storage method and the memory-reduction method are exactly the same, since the same paths are used to price the option. In our test, we choose the risk-free rate $r = 0.1$, the volatility $\sigma = 0.4$, and the expiration date $T = 0.5$ year. In Table 3.1, "CNM" stands for the results computed by the Crank-Nicolson method. The means and the standard deviations after 25 trials are shown under "Mean" and "STD" for both the memory-reduction method and the Brownian-bridge method. The two

Table 3.1: Black-Scholes model with $n = 10^5$ (50,000 plus 50,000 antithetic) and $m = 64$.

		Memory-reduction			Brownian- bridge		
S_0	CNM	Mean	STD	Error	Mean	STD	Error
6	4.0000	3.99220	0.00002	−0.00780	3.99220	0.00005	−0.00780
8	2.0951	2.09459	0.00192	−0.00051	2.09311	0.00226	−0.00199
10	0.9211	0.92117	0.00167	0.00007	0.92059	0.00232	−0.00051
12	0.3622	0.36190	0.00208	−0.00030	0.36181	0.00231	−0.00039
14	0.1320	0.13225	0.00125	0.00025	0.13184	0.00127	−0.00016

Table 3.2: CPU time in seconds and memory requirement when $S_0 = 10$.

m	32			32	64	128	Memory
n	20,000	40,000	80,000	20,000			requirement
Full-storage	4.25	8.59	17.19	4.25	8.50	16.98	$n(m+1)$
Memory-reduction	4.37	8.87	17.74	4.37	8.78	17.53	$m+2n$
Brownian-bridge	4.58	9.22	18.53	4.58	9.21	18.43	$n(\log_2 m+1)$

"Error" columns represent the difference between the corresponding "Mean" and "CNM". We observe that the accuracy is almost the same for all methods. Table 3.2 presents the average CPU times for five consecutive trials of each method. We see that our method brings about slight additional cost, but significantly reduces the storage requirement when compared with the other two methods. We also observe from Table 3.2 that, for all three methods there, the CPU time increases linearly with respect to m and n if either one is fixed. This is as expected, since the CPU times should be increasing like $O(mn)$.

3.5.2 Merton's jump-diffusion model

Merton's jump-diffusion process [39] can be described by the following stochastic differential equation under risk-neutral measure $\mathbb{Q}$ (generally not unique):

$$\frac{dS_t}{S_{t-}} = rdt + \sigma dW_t + dJ_t - \varpi dt. \tag{3.5}$$

Here $t-$ denotes the instant immediately before time t, $J_t = \sum_{k=1}^{N_t}(Y_k - 1)$ represents sudden jumps in price evolution, N_t is a Poisson counting process with intensity λ, and $\{\log Y_k\}_{k=1}^{N_t}$ are independent and identically distributed $\mathcal{N}(\alpha, \beta^2)$ numbers. Also in (3.5),

$$\varpi = \lambda \mathbf{E}^{\mathbb{Q}}[Y_k - 1] = \lambda \left[\exp\left(\alpha + \frac{1}{2}\beta^2\right) - 1\right] \tag{3.6}$$

is the compensator such that $\mathbf{E}^{\mathbb{Q}}[\exp(-rt)S_t] = S_0$. Rewriting (3.5) as (3.1), we have

$$L_t = -\frac{1}{2}\sigma^2 t + \sigma W_t + \sum_{k=1}^{N_t} \log(Y_k) - \varpi t. \tag{3.7}$$

Thus for Merton's jump-diffusion model, Step 2 in Algorithm 3.4.1 is

Algorithm 3.5.2 (Merton).

```
1. Generate $N_{i,j}$ ~Poisson($\lambda\Delta t$) using the inverse method [24, pp. 128]
2. Generate $Z^1_{i,j} \sim \mathcal{N}(0,1)$
3. If $N_{i,j} > 0$, generate $Z^2_{i,j} \sim \mathcal{N}(0,1)$
4. $\Delta L_{i,j} \leftarrow (-\frac{1}{2}\sigma^2 - \varpi)\Delta t + \sigma\sqrt{\Delta t}Z^1_{i,j} + \alpha N_{i,j} + \beta\sqrt{N_{i,j}}Z^2_{i,j}$
```

Now we test our method on an American put option under Merton's jump-diffusion model. The underlying stock price S_0 at the current time is \$40. The parameter values are $r = 8\%$, $\sigma = \sqrt{0.05}$, $\lambda = 5$, and $\beta = \sqrt{0.05}$. We let $\alpha = -\frac{1}{2}\beta^2$ such that $\mathbf{E}^{\mathbb{Q}}[Y_i] = 1$. The numerical results are reported in Table 3.3, where the columns "Mean" and "STD" are the means and the standard deviations

Table 3.3: Merton's model with $n = 10^5$ and $m = T/0.01$.

Strike K	Amin's	Mean	STD	Error
Expiring time $T = 0.25$ year				
30	0.674	0.6741	0.0064	0.0001
35	1.688	1.6872	0.0121	-0.0008
40	3.630	3.6248	0.0174	-0.0052
45	6.734	6.7288	0.0256	-0.0052
50	10.696	10.6867	0.0203	-0.0093
Expiring time $T = 1$ year				
30	2.720	2.7191	0.0132	-0.0009
35	4.603	4.6064	0.0204	0.0034
40	7.030	7.0242	0.0199	-0.0058
45	9.954	9.9461	0.0326	-0.0079
50	13.318	13.3050	0.0326	-0.0130

Table 3.4: CPU time in seconds and memory requirement when $T = 1, K = 40$.

m	50			50	100	200	Memory
n	20,000	40,000	80,000	20,000			requirement
Full-storage	22.05	43.86	87.77	22.05	43.52	86.88	$n(m+1)$
Memory-reduction	36.93	73.04	146.35	36.93	73.14	146.06	$m+2n$

obtained after 25 trials. We use the 200-time-step discrete time binomial tree model in [4] as a benchmark, and it is listed under the heading "Amin's". We observe that the two methods agree up to 2 decimals. Table 3.4 gives the average CPU times for five consecutive runs of the methods. Again the CPU time by our method is always less than twice of that by the full-storage method.

3.5.3 Variance gamma model

A variance gamma (VG) process [35] with parameters $\mu \in \mathbb{R}$, $\sigma > 0$, and $\nu > 0$ can be represented as a time-changed Brownian motion. Let $B_t = \mu t + \sigma W_t$ be a Brownian motion with drift μ and volatility σ. Define a gamma process G_t with independent gamma increments of mean h and variance νh over any non-overlapping time intervals of length h, or $G_t \sim \gamma(\frac{t}{\nu}, \nu) \sim \nu\gamma(\frac{t}{\nu})$. Then the three-parameter VG process X_t is defined by $X_t = B_{G_t}$ and its characteristic function is

$$\Phi_{X_t}(u) = \mathbf{E}[\exp(iuX_t)] = \left(\frac{1}{1 - i\mu\nu u + \frac{1}{2}\sigma^2\nu u^2}\right)^{\frac{t}{\nu}}. \tag{3.8}$$

Accordingly, the asset price process S_t is modeled as

$$S_t = S_0 \exp((r-q)t + X_t - \varpi t) \tag{3.9}$$

under the risk-neutral measure $\mathbb{Q}$ (generally not unique) with a continuous dividend yield of q and a constant continuously compounded interest rate of r. In model (3.9), the risk-neutral drift rate is $r - q$ and the compensator ϖ satisfies $\exp(\varpi) = \mathbf{E}^{\mathbb{Q}}[\exp(X_t)]$ such that $\mathbf{E}^{\mathbb{Q}}[\exp(-(r-q)t)S_t] = S_0$. By evaluating $\Phi_{X_t}(u)$ at $-i$, we have

$$\varpi = -\frac{1}{\nu}\log\left(1 - \mu\nu - \frac{1}{2}\sigma^2\nu\right). \tag{3.10}$$

Thus Step 2 in Algorithm 3.4.1 becomes:

Algorithm 3.5.3 (variance gamma).

1. *Generate* $Z_{i,j} \sim \mathcal{N}(0,1)$
2. *Generate* $\Delta G_{i,j} \sim \gamma(\frac{\Delta t}{\nu})$ *using Best's generator given in Algorithm 3.7.1*
3. $\Delta L_{i,j} \leftarrow \mu\Delta G_{i,j} + \sigma\sqrt{\Delta G_{i,j}}Z_{i,j} - \varpi\Delta t$

Table 3.5: Variance gamma model with $n = 10^5$ and $m = 56$.

Strike K	PIDE	Mean	STD	Error
1200	35.530	35.363	0.288	-0.167
1260	48.798	48.642	0.306	-0.156
1320	65.991	65.850	0.404	-0.141
1380	87.991	87.777	0.345	-0.214

Table 3.6: CPU time in seconds and memory requirement when $K = 1320$.

m	50			50	100	200	Memory
n	20,000	40,000	80,000	20,000			requirement
Full-storage	58.61	117.41	234.93	58.61	118.58	240.12	$n(m+1)$
Memory-reduction	112.53	225.34	450.73	112.53	229.05	462.36	$m+2n$

Now consider an American put option with maturity $T = 0.56164$ written on a stock with current price $S_0 = 1369.41$. The VG parameters after model calibration are given by $r = 0.0541$, $q = 0.012$, $\sigma = 0.20722$, $\nu = 0.50215$, and $\theta = -0.22898$. We test our method on various strike prices K and with $m = 56 \approx T/0.01$. The results are presented in Table 3.5. For comparison, results obtained by the partial integro-differential equation approach in [27] are given under "PIDE". As usual, the "Mean" and "STD" are the means and the standard deviations respectively, obtained after 25 trials. The difference between "Mean" and "PIDE" are computed in the column "Error". Again, the numerical results confirm the accuracy of our method. The average CPU times of five consecutive trials are given in Table 3.6, and the CPU time by our method is again bounded above by twice that by the full-storage method.

3.5.4 Remarks on the efficiency of the memory-reduction method

In the above three subsections, we have illustrated how to apply our memory-reduction method to specific exponential Lévy models. For both the full-storage method and the memory-reduction method, the computational cost is composed of two parts: the cost in path simulation and the cost in price computation. Compared with the full-storage method, the cost in path simulation is almost doubled in the memory-reduction method while the cost in price computation of both methods are the same. Hence our method always uses less than twice the time required by the full-storage method. In the following, we mention two factors affecting this overhead cost.

In Table 3.7, we give the ratio of the timing between the two methods in the "Ratio" rows for $m = 50$ and $n = 20,000$. In the table, the number in the square bracket $[\cdot]$ for each model is the average CPU time in seconds for generating 1,000 sample paths with 50 time steps. We observe from the table that the cost in path simulation in the Black-Scholes model is much less than that in the variance gamma model. As a consequence, our memory-reduction method almost produces no additional computational cost in the Black-Scholes model, while in the variance gamma model the CPU time of our method nearly doubles that of the full-storage method.

Another factor is the number of $S_{i,j}$ that are in-the-money. The rows "In-the-money (%)" in Table 3.7 count the average percentages of those "in-the-money" $S_{i,j}$ in the $m \cdot n$ samples in 5 trials. As the difference $K - S_0$ goes up, the number of "in-the-money" samples goes up, which leads to an increase in the cost of price computation. Consequently, the ratio goes down.

Table 3.7: CPU time in seconds with $m = 50$, $n = 20,000$.

Black-Scholes model [0.0331]

S_0	6	8	10	12	14
"In-the-money" (%)	98.9	87.3	49.0	15.5	4.7
Full-storage	13.8	11.52	6.68	2.74	1.48
Memory-reduction	14.11	11.78	6.89	2.87	1.61
Ratio	1.022	1.023	1.031	1.047	1.088

Merton's model ($T = 1$) [1.62]

Strike K	30	35	40	45	50
"In-the-money" (%)	21.6	33.6	52.4	69.7	79.1
Full-storage	17.94	19.37	21.65	23.71	24.87
Memory-reduction	32.31	33.86	36.02	38.11	39.29
Ratio	1.801	1.748	1.664	1.607	1.580

Variance gamma model [3.85]

Strike K	1200	1260	1320	1380
"In-the-money" (%)	11.8	16.3	23.1	37.2
Full-storage	57.90	58.51	59.37	61.04
Memory-reduction	112.47	113.18	113.91	115.61
Ratio	1.942	1.934	1.919	1.894

3.6 Conclusion

In this chapter, we propose a new simulation technique for pricing American options under exponential Lévy processes. It reduces the storage requirement to $O(m + n)$. For machines with limited memory, we can now enlarge m and n to improve the accuracy of the pricing. Furthermore, our memory-reduction method can easily be extended to pricing other path-dependent options with early-exercise features, such as Asian Bermudan options or multi-asset American options. Hence our method can be valuable in investigating option prices, especially those written on single or multiple assets with complex American triggers, long-term options, or any combination of these properties. We also remark that our memory reduction method has a natural extension to other relevant models such as stochastic volatility models, as long as the forward-path method (with no memory reduction) uses pseudorandom numbers in Monte Carlo simulation. However, the implementation becomes somehow more subtle, as different levels of randomness arise. We plan to consider such extensions in our future work.

3.7 Appendix

For completeness, here we give the algorithm for generating the gamma random variables. We also give the commands in FORTRAN and MATHEMATICA for finding the seeds of a sequence of random numbers.

Algorithm 3.7.1 below generates Gamma random variables $\gamma(a)$ with density

$$p(x) = \frac{x^{a-1}}{\Gamma(a)}e^{-x}$$

when $a \geq 1$. For $a < 1$, one uses the transformation $\gamma(a) = \gamma(1+a)U^{1/a}$ with $U \sim \mathcal{U}[0,1]$. See [17, pp. 410 and pp. 420] for a comprehensive discussion.

Algorithm 3.7.1 (Best's generator).

```
1. $b \leftarrow a-1$, $c \leftarrow 3a - \frac{3}{4}$
Repeat
    2. Generate random variables $U, V \sim \mathcal{U}[0,1]$
    3. $W \leftarrow U(1-U)$, $Y \leftarrow \sqrt{\frac{c}{W}}(U - \frac{1}{2})$, $X \leftarrow b + Y$
    4. If $X < 0$, go to Repeat
    5. $Z \leftarrow 64W^3V^2$
Until $\log(Z) \leq 2b\log(\frac{X}{b} - Y)$
Return $X$
```

In FORTRAN 90, the command to get a $\mathcal{U}[0,1]$ number is `rand()`. The commands to set the seed to `d` are:

```
call random_seed(size=k)
seed(1:k)=d
call random_seed(put=seed(1:k))
```

where `k` is the number of 32-bit words used to hold the seed. The commands to extract the current seed `d` are:

```
call random_seed(get=current(1:k))
d=current(1:k)
```

In MATHEMATICA, the seeds are set by "`SeedRandom[d]`". To extract the current seed, use "`c=$RandomState`". MATHEMATICA provides $\mathcal{U}[0,1]$ numbers with the command "`Random[ ]`".

Bibliography

[1] `http://www.cuhk.edu.hk/itsc/compenv/research-computing/organon/`.

[2] `http://www.ll.mit.edu/MatlabMPI/`.

[3] A. Almendral and C. W. Oosterlee. Numerical valuation of options with jumps in the underlying. *Appl. Numer. Math.*, 53:1–18, 2005.

[4] K. I. Amin. Jump diffusion option valuation in discrete time. *Journal of Finance*, 48:1833–1863, 1993.

[5] L. Andersen and J. Andreasen. Jump-diffusion processes: volatility smile fitting and numerical methods for pricing. *Review of Derivatives Research*, 4:231–262, 2000.

[6] A. Bermon and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* SIAM, 1994.

[7] N. H. Bingham and R. Kiesel. *Risk-Neutral Valuation: Pricing and Hedgeing of Financial Derivatives.* Springer-Verlag, 2nd edition, 2004.

[8] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[9] P. P. Boyle. Option: a Monte Carlo approach. *Journal of Financial Economics*, 4:323–338, 1977.

[10] R. H. Chan, Y. Chen, and K. M. Yeung. A memory reduction method in pricing American options. *Journal of Statistical Computation and Simulation*, 74:501–511, 2004.

[11] R. H. Chan, C. Y. Wong, and K. M. Yeung. Pricing multi-asset American-style options by memory reduction Monte Carlo methods. *Applied Mathematics and Computation*, 179:535–544, 2006.

[12] S. K. Chaudhary. American options and the LSM algorithm: quasi-random sequences and Brownian bridges. *The Journal of Computational Finance*, 8:101–115, 2005.

[13] R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1:1–14, 2001.

[14] R. Cont and P. Tankov. *Financial Modelling with Jump Processes.* Chapman & Hall/CRC, 2004.

[15] R. Cont and E. Voltchkova. A finite difference scheme for option pricing in jump diffusion and exponential Lévy models. *SIAM J. Numer. Anal.*, 43:1596–1626, 2005.

[16] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systems. *J. Comput. Phys.*, 176:430–455, 2002.

[17] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, 1986.

[18] Y. d'Halluin, P. A. Forsyth, and K. R. Vetzal. Robust numerical methods for contingent claims under jump diffusion processes. *IMA J. Numer. Anal.*, 25:87–112, 2005.

[19] R. J. Elliott and P. E. Kopp. *Mathematics of Financial Markets.* Springer-Verlag, 2nd edition, 2005.

[20] L. Feng and V. Linetsky. Pricing options in jump-diffusion models: an extrapolation approach. *Operations Research*, 56:304–325, 2008.

[21] I. P. Gavrilyuk and V. L. Makarov. Algorithms without accuracy saturation for evolution equations in Hilbert and Banach spaces. *Math. Comp.*, 74:555–583, 2004.

[22] I. P. Gavrilyuk and V. L. Makarov. Exponentially convergent algorithms for the operator exponential with applications to inhomogeneous problems in Banach spaces. *SIAM J. Numer. Anal.*, 43:2144–2171, 2005.

[23] I. P. Gavrilyuk and V. L. Makarov. An exponentially convergent algorithm for nonlinear differential equations in Banach spaces. *Math. Comp.*, 76:1895–1923, 2007.

[24] P. Glasserman. *Monte Carlo Methods in Financial Engineering.* Springer-Verlag, 2003.

[25] J. M. Harrison and S. R. Pliska. Martingale and stochastic integrals in the theory of continuous trading. *J. Econom. Theory*, 11:215–260, 1981.

[26] N. J. Higham. *Functions of Matrices: Theory and Computation.* SIAM, 2008.

[27] A. Hirsa and D. B. Madan. Pricing American options under variance gamma. *The Journal of Computational Finance*, 7:63–80, 2003.

[28] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, UK, 1994.

[29] A.-K. Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM J. Sci. Comput.*, 26:1214–1233, 2005.

[30] S. G. Kou. A jump-diffusion model for option pricing. *Management Science*, 48:1086–1101, 2002.

[31] J. Lee and D. Sheen. A parallel method for backward parabolic problems based on the Laplace transformation. *SIAM J. Numer. Anal.*, 44:1466–1486, 2006.

[32] F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation: a simple least-squares approach. *The Review of Financial Studies*, 14:113–147, 2001.

[33] M. López-Fernández and C. Palencia. On the numerical inversion of the Laplace transform of certain holomorphic mapping. *Appl. Numer. Math.*, 51:289–303, 2004.

[34] M. López-Fernández, C. Palencia, and A. Schädle. A spectral order method for inverting sectorial Laplace transforms. *SIAM J. Numer. Anal.*, 44:1332–1350, 2006.

[35] D. B. Madan, P. Carr, and E. C. Chang. The variance gamma process and option pricing. *European Finance Review*, 2:79–105, 1998.

[36] B. Mandelbrot. The variation of certain speculative prices. *Econometrica*, 31:757–758, 1963.

[37] A. Matache, C. Schwab, and T. Wihler. Fast numerical solution of parabolic integrodifferential equations with applications in finance. *SIAM J. Sci. Comput.*, 27:369–393, 2005.

[38] W. McLean, I. Sloan, and V. Thomée. Time discretization via Laplace transformation of an integro-differential equation of parabolic type. *Numer. Math.*, 102:497–522, 2006.

[39] R. C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3:125–144, 1976.

[40] B. V. Minchev and W. M. Wright. A review of exponential integrators for first order semi-linear problems. Technical Report 2/2005, Norwegian University of Science and Technology, 2005.

[41] C. Moler. *Numerical Computing with MATLAB*. SIAM, 2004.

[42] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31:1192–1201, 1988.

[43] S. C. Reddy. Pseudospectra of Wiener-Hopf integral operators and constant-coefficient differential operators. *J. Integral Equations Appl.*, 5:369–403, 1993.

[44] S. C. Reddy and L. N. Trefethen. Pseudospectra of the convection-diffusion operator. *SIAM J. Appl. Math.*, 54:1634–1649, 1994.

[45] C. Ribeiro and N. Webber. A Monte Carlo method for the normal inverse Gaussian option valuation model using an inverse Gaussian bridge. Technical report, City University, 2003.

[46] C. Ribeiro and N. Webber. Valuing path-dependent options in the variance gamma model by Monte Carlo with a gamma bridge. *The Journal of Computational Finance*, 7:81–100, 2004.

[47] T. Schmelzer. *The fast evaluation of matrix functions for exponential integrators*. PhD thesis, University of Oxford, 2007.

[48] D. Sheen, I. Sloan, and V. Thomée. A parallel method for time-discretization of parabolic prolems based on contour integral representation and quadrature. *Math. Comp.*, 69:177–195, 2000.

[49] D. Sheen, I. Sloan, and V. Thomée. A parallel method for time discretization of parabolic equations based on Laplace transformation and quadrature. *IMA J. Numer. Anal.*, 23:269–299, 2003.

[50] S. E. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models.* Springer-Verlag, 2004.

[51] A. Talbot. The accurate numerical inversion of Laplace transforms. *IMA J. Numer. Anal.*, 23:97–120, 1979.

[52] J. Toivanen. Numerical valuation of European and American options under Kou's jump-diffusion model. *SIAM J. Sci. Comput.*, 30:1949–1970, 2008.

[53] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators.* Princeton University Press, New Jersey, 2005.

[54] L. N. Trefethen, J. A. C. Weideman, and T. Schmelzer. Talbot quadratures and rational approximations. *BIT*, 46:653–670, 2006.

[55] J. A. C. Weideman. Optimizing Talbot's contours for the inversion of the Laplace transform. *SIAM J. Numer. Anal.*, 44:2343–2362, 2006.

[56] J. A. C. Weideman. Improved contour integral methods for parabolic PDEs. *IMA J. Numer. Anal.*, 30:334–350, 2010.

[57] J. A. C. Weideman and L. N. Trefethen. Parabolic and hyperbolic contours for computing the Bromwich integral. *Math. Comp.*, 76:1341–1356, 2007.

[58] P. Wilmott, S. Howison, and J. Dewynne. *The Mathematics of Financial Derivatives.* Cambridge University Press, 1998.