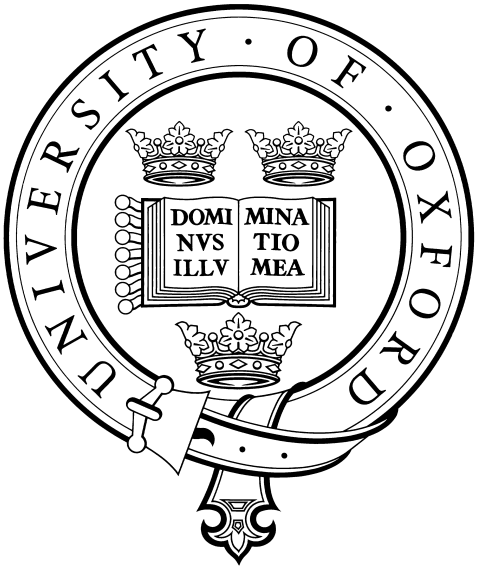
Jumping Hedges

Hedging Options Under Jump-Diffusion



Nicholas Hinde

Linacre College

University of Oxford

A thesis submitted in partial fulfillment of the requirements for the MSc in

Mathematical Modelling and Scientific Computing

September 2006

Acknowledgements

I would like to thank my supervisors, Professor Mike Giles and Dr Sam Howison for their wise advice throughout the course of writing this thesis. Professor Giles kindly offered me some of his own MATLAB code for pricing and the Greeks and he has made countless poignant observations on my numerical implementation, leading to improvements in the accuracy and speed of the programs. Dr Howison intelligently noticed that Hawkes processes could be applied to jump-diffusion.

I would also like to thank Alex Prideaux for spending considerable time working jointly with me to implement a numerical solver for the PIDE. I thank Marianne Bach for reviewing the thesis. I owe thanks to the Engineering and Physical Sciences Research Council (EPSRC) for funding during the MSc course. Finally, I thank my family for their unfaltering encouragement and support in whatever I do.

Abstract

Under Merton's jump-diffusion, we verify the series solution for European option prices through Monte Carlo estimation and numerically solving the partial-integro differential equation (PIDE). We extend the PIDE solver to American options through the penalty method, preserving quadratic convergence. Delta hedging a target option under jump-diffusion is inadequate, due to the jumps. We add short-dated options to our hedge, with weights chosen through Gauss-Hermite quadrature or least squares. The latter results in better replication of the target. We weight least squares using the transition PDF and the jump amplitude PDF. We find that the former performs well when semi-static hedging, even with several options. The latter performs better when frequently rebalancing. We refine jump-diffusion by replacing the Poisson process with Hawkes, which is tractable for pricing and simulating. We discover that mis-specifying the counting process barely affects hedging performance. A general rule is that the greater the number of hedging options, the more successful the hedge. We speculate that the hedge may be further improved by refining our choice of weighting function and optimizing over strikes and rebalance times in addition to weights.

Contents

List of Figures

Chapter 1

Introduction

1.1 Aim

On 19 October 1987 the Dow Jones industrial average dropped by 22.5% (Figure 1.1). Since there was good correlation between the major indices, the risk was not diversified through a well-balanced portfolio of investments. According to the conventional Black-Scholes market model, this one-day price drop should on average occur once every 10^6 years[1], or once every thousand millennia. However, crashes of a comparable scale occurred on both 12 December 1914 and on 28 October 1929. The conventional model massively underestimates the probability of a crash. Therefore, we ask:

- Is there a model that more accurately represents jumps in share price?
- Using such a model, how can a financial institution lessen its exposure to jumps?

We assume it is impossible to anticipate jumps in prices and instead aim to minimize the risk of a jump if it were to occur tomorrow or any day in the future.

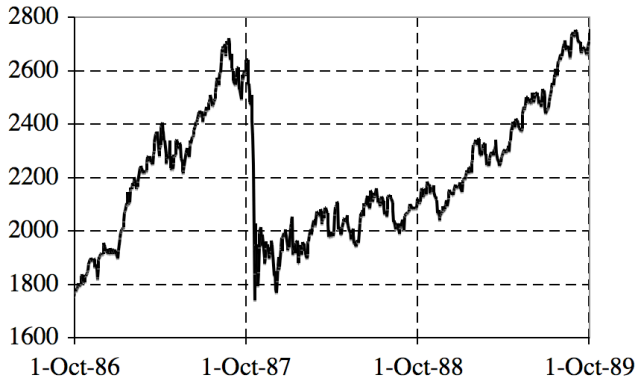


Figure 1.1: The price (US$) of the Dow Jones industrial average (DJIA) index

[1]See section A.1.1 for a detailed explanation.

1.2 Basic Concepts

Options are contracts for future cash-flow between two parties, derived from one or more observable quantities, the underlyings. The price of an option is the cost of neutralizing the risk to which the writer is exposed, or equivalently the cost of replicating the option. Such replication is called hedging. If the option can be replicated exactly, then the price is independent of risk preferences. However, if it cannot be perfectly hedged, risk preferences come into play. In that case, options prices in the market can be decomposed into the cost of hedging and a risk premium, the market price of the non-diversifiable risk.

Two fundamental types of options are European and American. Both contracts stipulate a payoff, representing the income for the buyer when each is exercised. If the future value of the underlyings is uncertain, then the future payoff of the option is also uncertain. However, the relationship between the value of the underlyings and the payoff of the option is explicitly stated in the contract.

A European option may only be exercised at expiry, but holders of American options may choose to exercise at *any* time prior to or at maturity. There are many other types of options, such as Asian and Bermudan, with different contractual obligations. We study European and American options where the payoff is a function of a single underlying, the asset price. To value such options we must determine a model for the asset price evolution.

The conventional model used by many market agents is that of Black-Scholes, first presented in [4]. We study an extension of Black-Scholes mapped out in [26], called Merton's jump-diffusion. In Merton's world, it is impossible to perfectly hedge an option by buying and selling other instruments and hence the market is incomplete. However, we demonstrate that a straightforward least squares approach provides a very good hedge. The success of the hedge is found to be fairly insensitive to misspecification of the model.

1.3 Organization

This paper is divided into three major parts: chapter 2 incrementally builds the jump-diffusion framework and related tools; chapter 3 explains and simulates hedging strategies; chapter 4 studies Hawkes processes as an extension to jump-diffusion.

We begin by introducing Merton's jump-diffusion model in section 2.1. To evaluate the effectiveness of hedging strategies, we must create sample asset price paths. Section 2.2 explains how to do so by applying Monte Carlo simulation to jump-diffusion. Merton's model for asset price evolution may be written as a differential equation with an integral term, a partial-integro differential equation (PIDE). In order to price American options and create exercise boundaries, section 2.6 presents a scheme for numerically solving the PIDE.

Having laid the foundations for hedging, in section 3.1 we introduce the concept of an incomplete market and the consequences for trying to eliminate risk. We discuss two alternative hedging strategies: semi-static and dynamic. The former replicates the value of the target option at some future time, whilst the latter regularly adjusts the hedge in an attempt to move hand-in-hand with the option price. We study two different option replication techniques: Gauss-Hermite quadrature and least squares. Simulations test the effectiveness of both the replication techniques and hedging strategies.

Calibrating Merton's jump-diffusion to market option prices may yield a variety of plausible parameter sets, and therefore the possibility of model mis-specification is strong. In particular, we may miscalculate correlation between jumps, which can be described by a Hawkes process. Chapter 4 discusses this process and in the case of exponential decay for the memory, some useful properties are derived. We acknowledge that analytical formulae for option prices under jump-diffusion with a Hawkes process should be forthcoming.

Using the numerical results of chapter 3, we conclude by comparing hedging strategies in chapter 5. In particular, the suitability of each strategy to different types of options is discussed. In addition, we comment on the choice of hedging instruments, strikes and rebalance times. We also comment on the relevance of Hawkes processes and the implications for pricing and hedging, given the results in chapter 4. Finally, we foresee opportunities for future research.

Chapter 2

Valuation

2.1 Jump-Diffusion

2.1.1 Asset Price Evolution with Jumps

In the Black-Scholes model, an asset price moves under geometric Brownian motion with drift, a purely diffusive evolution. Given a terminal payoff at time T, the price of a European option at time t ($t < T$) is the solution of a linear parabolic partial differential equation (PDE), which can be transformed into the backwards heat equation. This yields a straightforward analytical form for the price of European vanilla options, given in [19] (p.48). Similarly, the derivatives with respect to the model parameters, called the Greeks, also have closed forms. For the purpose of tractibility, it is preferable that any other asset price model also has analytical solutions for European vanilla options.

There is evidence in the market that geometric Brownian motion with drift is not an accurate model for asset prices. The Black-Scholes European option pricing formula is a bijective map from prices to volatility, the standard deviation of returns. The Black-Scholes model assumes a constant volatility, but market prices map to an "implied volatility smile", which is non-constant. Equivalently, there is more weight in the tails of the returns distribution than in a Gaussian distribution. This is clearly evident in Figure 2.1, comparing actual market returns with the returns of a simulated price path: the outliers occur far too infrequently in the Black-Scholes model. Two popular models that try to right these imperfections are stochastic volatility and models with jumps. A pure jump model, such as variance-gamma [20] (p.564), does not incorporate diffusion. However, diffusive models do capture *most* of the behaviour of asset prices, so we choose to study Merton's mixed jump-diffusion [26].

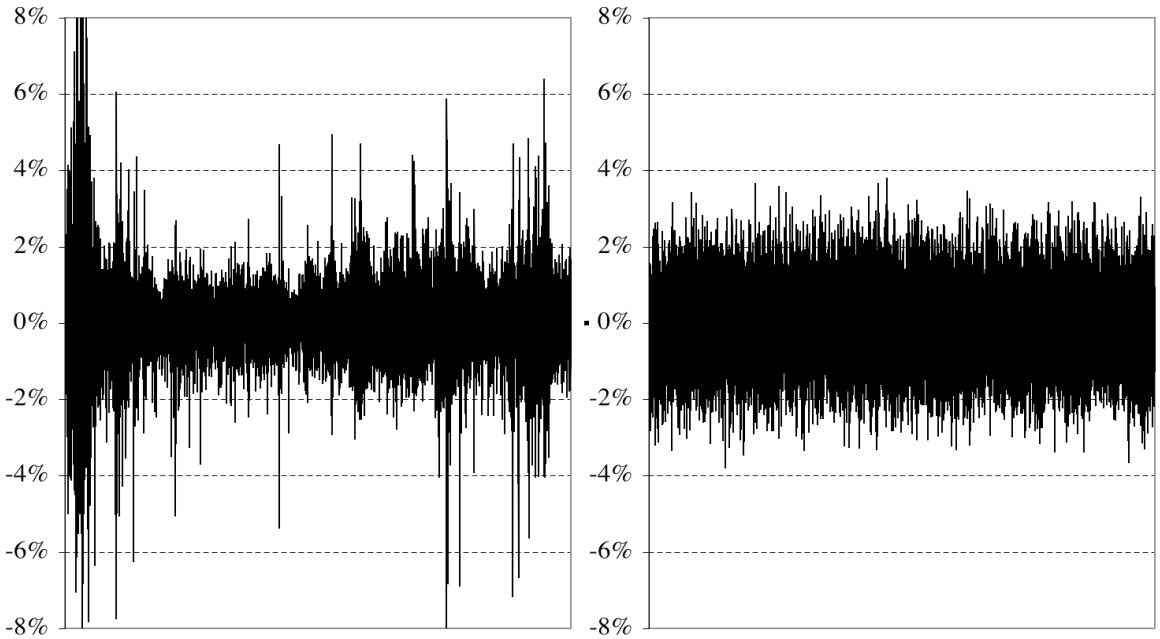


Figure 2.1: Daily returns of the DJIA from January 1930 to August 2006 on the left and 76.5 years of Brownian motion with identical volatility on the right.

In jump-diffusion, the evolution is a combination of geometric Brownian motion and discontinuities, or jumps. The stochastic differential equation (as in [26]) is

$$\frac{\mathrm{d}S_t}{S_{t-}} = \upsilon \mathrm{d}t + \sigma \mathrm{d}W_t + \mathrm{d}J_t. \tag{2.1}$$

where the denominator of the left hand side is

$$S_{t-} = \lim_{u \uparrow t} S(u)$$

to ensure that the jump-multiplier is non-anticipating. It is more convenient to work in a transformed variable $x = \log S$, which means jumps are additive, rather than multiplicative. Under this transformation, (2.1) becomes ([7], p.111)

$$x_t = \omega t + \sigma W_t + \sum_{j=1}^{N(t)} \log Y_j \tag{2.2}$$

where $\omega = \upsilon - \frac{1}{2}\sigma^2$. The total jump multiplier for a period of time $[0, t]$ is ([14])

$$J_t = \sum_{j=1}^{N(t)} (Y_j - 1)$$

where $N(t)$ is a counting process satisfying $N(t) = \sup\{n : \tau_n \leq t\}$ and each τ_j is a jump time, with Y_j the corresponding jump amplitude. Each jump multiplies the asset price instantaneously before the jump, so $x_{\tau_j} = x_{\tau_j -} + \log Y_j$. The process W_t in (2.2) represents Brownian motion with mean zero and standard deviation $\sqrt{t}$.

2.1.2 Merton's Jump-Diffusion Model

In Merton's jump-diffusion model, J is a compound Poisson process and hence $N(t)$ is a Poisson counting process. The probability that there are n jumps in the time period $[0, t]$ is

$$\mathbb{P}[N(t) = n] = \frac{(\lambda t)^n}{n!} e^{-\lambda t}. \tag{2.3}$$

This models the occurrence of unpredictable events where the expectation is known: $\mathbb{E}[N(t)] = \lambda t$. The jump amplitudes are assumed to be independent and identically distributed (i.i.d), and in Merton's model they are log-normally distributed:

$$\log Y_j \sim \mathcal{N}(\mu, \delta^2).$$

Overall, the parameters of Merton's jump-diffusion are:

$r =$ the risk-free rate;

$\upsilon =$ the coefficient of drift of the diffusive process in terms of S;

$\sigma =$ the coefficient of volatility of the diffusive process in terms of S;

$\lambda =$ the jump intensity;

$\mu =$ the mean jump amplitude in terms of x;

$\delta =$ the standard deviation of the jump amplitude in terms of x.

The expectation of the price change due to a jump is $\kappa = \mathbb{E}(Y_j - 1) = \exp(\mu + \frac{1}{2}\delta^2) - 1$, because the probability density function (PDF) of the jump amplitude in S is

$$g(Y) = \frac{\exp\left[-\frac{(\log Y - \mu)^2}{2\delta^2} \right]}{\sqrt{2\pi}\delta Y}. \tag{2.4}$$

In terms of x, the diffusion and each jump are i.i.d normal random variables. The convolution of two Gaussian distributions is also a Gaussian distribution. Therefore, we can express the transition probability density function (PDF) for the random variable x_t as a sum of each Gaussian PDF for n jumps with weights given by the probability that $N(t) = n$ ([7], p.111):

$$p_t(x) = e^{-\lambda t} \sum_{n=0}^{\infty} \frac{(\lambda t)^n \exp\left[-\frac{(x - \omega t - n\mu)^2}{2\sigma^2 t + n\delta^2} \right]}{n! \sqrt{2\pi(\sigma^2 t + n\delta^2)}}. \tag{2.5}$$

We plot this formula for the transition PDF against the distribution resulting from a large number of simulated asset price evolution paths in section 2.2.4.

We use parameter values detailed in Table 2.1 for Merton's jump-diffusion as given in [17], obtained via calibration with market option prices for an equities index and are therefore claimed to be realistic. [1] also employs very similar parameter values.

Parameter	**Value**
r	0.05
σ	0.2
λ	0.1
μ	-0.92
δ	-0.425

Table 2.1: Market calibrated parameters.

Unless otherwise stipulated, all numerical examples and plots throughout this paper use these parameters values. Our choice of μ and δ imply that the mean jump price change is $\kappa = -0.5638$ (to 4 significant figures). However, if the parameters are calibrated to assets in other markets, such as foreign exchange, κ may be nearer zero or even positive.

2.1.3 Analytical Series Solution for Jump-Diffusion

The transition PDF (2.5) is a weighted sum of Black-Scholes PDFs where the parameters reflect the n jumps that occur over the time-frame. Therefore, the Black-Scholes European vanilla option formulae can be translated into a series solution for jump-diffusion. In order to make $S_t e^{-rt}$ a martingale, the drift parameter must be set to $\upsilon = r - \lambda\kappa$, defining the risk-neutral numeraire. Therefore, as detailed in [21] (p.345) the price of a European option under jump-diffusion is

$$\begin{aligned} V(S,T) &= \sum_{n=0}^{\infty} \frac{\left(\bar{\lambda}T\right)^n}{n!} e^{-\bar{\lambda}T} \mathrm{BS}(S_0, \sigma_n, r_n, T, K), \\ \text{where} \qquad \sigma_n &= \sqrt{\sigma^2 + n\delta^2/T}, \\ r_n &= r - \lambda\kappa + n\log(1+\kappa)/T, \\ \bar{\lambda} &= \lambda(1+\kappa). \end{aligned}$$

Figure 2.2 shows graphically the value and three sensitivities, known as Greeks, for the European call and put options:

$$\Delta = \frac{\partial V}{\partial S}, \qquad \Gamma = \frac{\partial^2 V}{\partial S^2}, \qquad \text{vega} = \frac{\partial V}{\partial \sigma}.$$

Under jump-diffusion, we may also compute sensitivities with respect to λ, μ and δ.

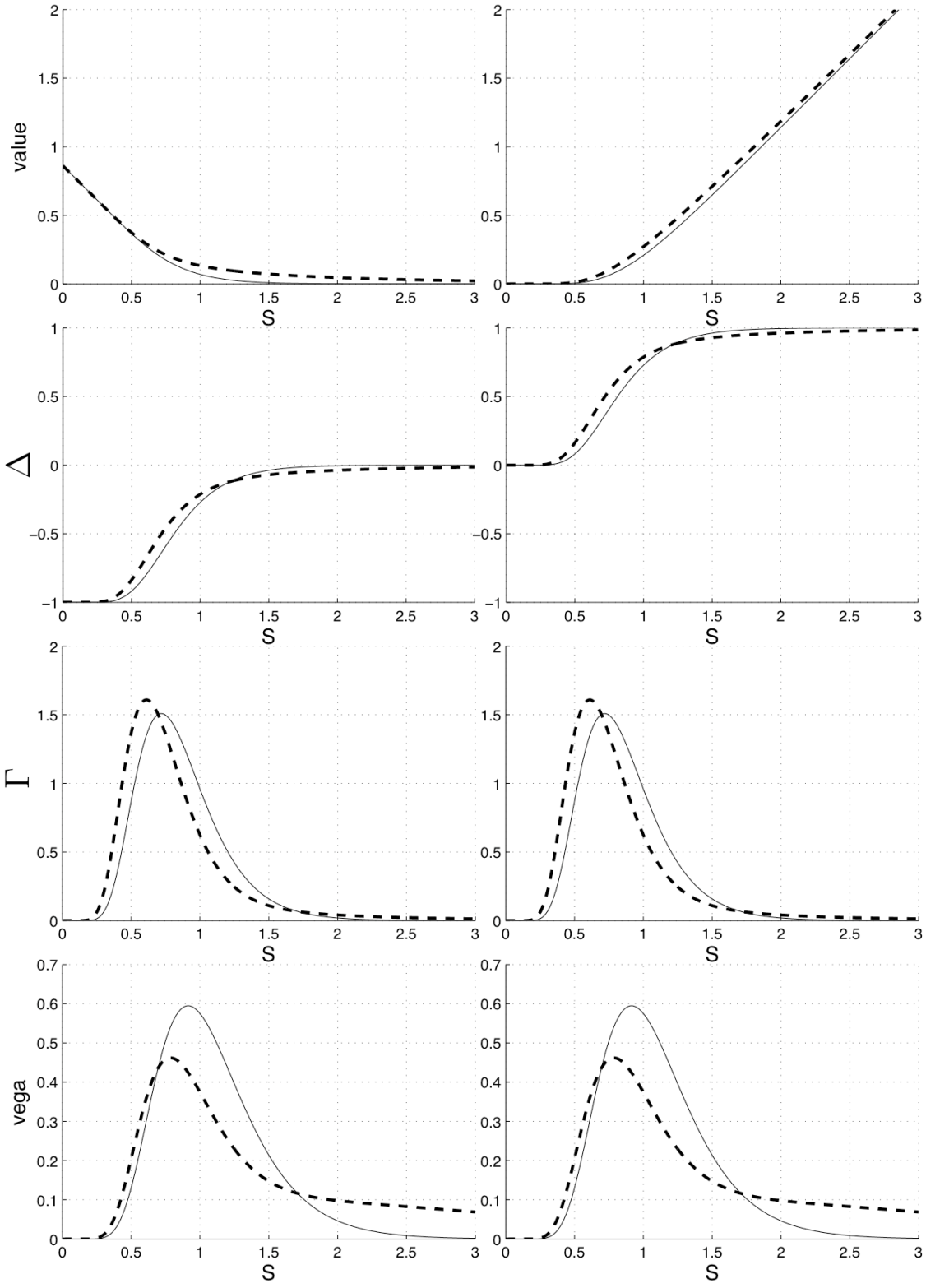


Figure 2.2: The Black-Scholes analytical formulae (continuous lines) against the Merton analytical formulae (broken line) for a European put on the left and European call on the right. The maturity is $T = 3$, which is fairly large so that the Black-Scholes and Merton curves are clearly distinguishable.

As illustrated in Figure 2.2, the Black-Scholes values of Γ and vega are the same for a call and a put; this symmetry also holds under jump-diffusion. The value of a vanilla option under jump-diffusion is greater than or equal to that under Black-Scholes, because jumps effectively increase the volatility of the underlying ([21], p.351). As $S \to 0$ and as $S \to \infty$, the Black-Scholes and Merton formulae approach one another. This is not obvious in the vega plot, due to the large value of T we have chosen.

In the Black-Scholes framework, the American call price is equal to the European call price, but the put prices differ; the same is true in Merton's world. However, the American put does not have an analytical solution under either model. To value American puts, we may numerically solve a partial integro-differential equation (PIDE) with a constraint to reflect the possibility of early exercise.

2.1.4 PIDE Derivation

Letting $\tau = T - t$ the Black-Scholes partial differential equation (PDE) becomes a partial-integro differential equation (PIDE) under jump-diffusion ([17], p.4):

$$V_\tau = \frac{\sigma^2 S^2}{2} V_{SS} + (r - \kappa\lambda) S V_S - rV + \lambda \left(\int_0^\infty V(S\eta, \tau) g(\eta) \mathrm{d}\eta - V(S, \tau) \right) \tag{2.6}$$

where $g(\eta)$ is the PDF for the jump amplitude η and the subscripts mean partial differentiation, e.g. $V_S = \partial V / \partial S$. Let $y = \log(\eta)$ and the integral in 2.6 is ([11], p.5)

$$\int_0^\infty V(S\eta, \tau) g(\eta) d\eta = \int_{-\infty}^\infty \bar{V}(x + y) \bar{f}(y) \mathrm{d}y \tag{2.7}$$

which is a convolution integral. In Merton's model, $\bar{f}(y)$ is the Gaussian PDF.

It is more convenient to write the PIDE in terms of $x = \log S$. Without loss of generality we may assume that $S_0 = 1$ and then the transformation [7] (p.388)

$$u(\tau, x) = e^{r\tau} V(\tau, e^x) \tag{2.8}$$

results in the PIDE

$$\frac{\partial u}{\partial \tau}(\tau, x) = \frac{\sigma^2}{2} \frac{\partial^2 u}{\partial x^2}(\tau, x) + \left(r - \frac{\sigma^2}{2} \right) \frac{\partial u}{\partial x}(\tau, x) +$$
$$\lambda \int g(z) \left[u(\tau, x + z) - u(\tau, x) - (e^z - 1) \frac{\partial u}{\partial x}(\tau, x) \right] \mathrm{d}z \tag{2.9}$$

which is valid $\forall\ (\tau, x) \in [0, T] \times \mathbb{R}$. The discounting (2.8) removes the rV term from (2.6), simplifying the resultant PIDE slightly. Letting $H(S)$ be the option payoff, the initial condition is $u(0, x) = H(e^x)\ \forall\ x \in \mathbb{R}$. For example, $H(S) = (S - 1)^+$ for a call option and $H(S) = (1 - S)^+$ for a put option. We solve (2.9) approximately using numerical techniques in section 2.3.

2.2 Monte Carlo Methods

2.2.1 Monte Carlo Fundamentals

Consider estimating an integral over a continuous set $\mathbb{X}$:

$$F = \int_{\mathbb{X}} f(x)\mathrm{d}x \tag{2.10}$$

where no closed form solution F exists. If $\mathbb{X}$ is one-dimensional, we may use an integration rule to approximate (2.10) by projecting $f(\cdot)$ onto a discrete domain. For example, the trapezium rule gives

$$F \approx \frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{N-1} f(x_i) + f(x_N)\right)$$

where $\mathbb{X}$ is approximated by $\{x_0, x_1, ..., x_N\}$, the interval $h = x_{n+1} - x_n$ is constant and N is the number of integration nodes. If $\mathbb{X}$ is bounded below and above by $a < x < b$, the lower and upper truncation of $\mathbb{X}$ is natural. However, if $\mathbb{X}$ is unbounded, e.g. $\mathbb{X} = \mathbb{R}$, then this choice is less straightforward.

Alternatively, consider the integral

$$\tilde{F} = \int_{\mathbb{X}} f(x)w(x)\mathrm{d}x. \tag{2.11}$$

where $w(x)$ is a positive weighting function satisfying $\int_{\mathbb{X}} w(x)\mathrm{d}x = 1$. For example, if $w(x)$ is a probability density function (PDF), then $\tilde{F} = \mathbb{E}[f(x)]$. To approximate $\tilde{F}$ using an integration rule, nodes x_n and weights w_n should be chosen appropriately. If $\mathbb{X}$ is multi-dimensional, the choice of such a rule, x_n and w_n may significantly affect the accuracy of the approximation. Furthermore, implementing such a scheme may be tedious.

An alternative is Monte Carlo methods. Crudely, this is throwing darts in a random direction and inferring a dartboard's area from the proportion that hit it. Mathematically, the idea is to take a random sample, evaluate the function at each sample point and take the average. As the sample size tends to infinity, the average should tend to the exact solution. More concretely, in the case of (2.10), with a uniform distribution on $\mathbb{X}$ given by $\mathcal{U}[\mathbb{X}]$, a Monte Carlo estimate is F_M,

$$F \approx F_M = \frac{1}{M}\sum_{i=1}^{M} f(x_i) \qquad \text{where each } x_i \text{ is a random draw from } \mathcal{U}[\mathbb{X}].$$

For the weighted integral (2.11), let $W^{-1}(x) : [0,1] \mapsto \mathbb{X}$ be the inverse of $W(x) = \int_0^x w(y)dy$, then the Monte Carlo estimate is $\tilde{F}_M$,

$$\tilde{F} \approx \tilde{F}_M = \frac{1}{M}\sum_{i=1}^{M} f\big(W^{-1}(y_i)\big) \qquad \text{where each } y_i \text{ is a random draw from } \mathcal{U}[0,1].$$

The quantity $\tilde{F}_M$ is a random variable, a function of M uniformly distributed random numbers, that converges to the exact solution almost surely,

$$\lim_{M\to\infty} \tilde{F}_M = \tilde{F}$$

and similarly for F_M with respect to F. Let the error in the estimate be defined as

$$\epsilon_M[f] = \tilde{F}_M - \tilde{F}.$$

Provided $W(x)$ and f both have a finite second moment, by the Central Limit theorem,

$$\lim_{M\to\infty} \epsilon_M[f] = \frac{\sigma_f}{\sqrt{M}} Z \qquad \text{where } Z \sim \mathcal{N}(0,1)$$

and σ_f is the standard deviation of f. This gives the order of convergence for Monte Carlo methods in general. Note that one-dimensional integral rules such as the trapezium and Simpson's, have faster convergence. However, in higher dimensions, Monte Carlo is guaranteed to converge in most circumstances and is often quicker.

2.2.2 Applying Monte Carlo to Pricing

Although Monte Carlo simulation is comparatively slow for pricing options on a single asset, it can easily be extended to many dimensions. In addition, it provides a convenient framework with which to approximate jumps in asset price and to evaluate hedging errors, as we shall see in chapter 3.

To price European options by Monte Carlo, we follow a simple recipe ([14] p.30), independent of the asset price evolution model:

1. Simulate M paths and obtain the terminal asset price S_T for each;
2. Compute the discounted payoff for each S_T;
3. Average over all paths.

However, to price American options, which are path dependent, we must approximate using a Bermudan option, with a finite number N of exercise times. If the path is in the exercise region at any of these times, then the option is exercised. Hence, the Monte Carlo procedure is more involved:

1. Discretize time so that exercise may only occur at $t \in \mathbb{T} = \{t_i : i \in \{0, 1, ..., N\}\}$;
2. Simulate M paths and obtain the asset price $S_t \ \forall \ t \in \mathbb{T}$;
3. For each path compute the discounted payoff at time $t_k = \inf\{t_i \in \mathbb{T} : S_{t_i} \in \mathbb{S}\}$ where $\mathbb{S}$ is the exercise region;
4. Average over all paths.

Here we have implicitly assumed that the $\mathbb{S}$ is known a priori. In the case of jump-diffusion with constant parameters, the exercise boundary may indeed be computed prior to Monte Carlo simulation, e.g. by solving the PIDE, as in section (2.6). Other techniques for evaluating American options using Monte Carlo are complicated to implement and often make further approximating assumptions.

To compute the sensitivities (the Greeks), we can perturb the relevant parameter and use the same sample paths. In doing so, any bias in one path is almost the same in the perturbed case and hence it should cancel out. Two common techniques are finite differences and using a complex variable for the perturbation. For example, the option's delta may be computed through the central difference

$$\Delta \approx \frac{V(S + \mathrm{d}S) - V(S - \mathrm{d}S)}{2\mathrm{d}S}. \tag{2.12}$$

Let Δ_{MC} be the Monte Carlo estimate to Δ. The error is

$$|\Delta_{\text{exact}} - \Delta_{MC}| = \left| \frac{\left(\sigma_{V(S+\mathrm{d}S)} - \sigma_{V(S-\mathrm{d}S)}\right) Z}{\sqrt{M}} \right| \approx \left| \frac{\left(\sigma_{+\mathrm{d}SV(S)} - \sigma_{-\mathrm{d}SV(S)}\right) Z}{\sqrt{M}} \right|. \tag{2.13}$$

where the same sample paths S are used for both $S + \mathrm{d}S$ and $S - \mathrm{d}S$, and Z is a random variable with standard normal distribution. This error may be much smaller than the error if two independent Monte Carlo paths are used. To see this, let the two paths be S^1 and S^2. Then the difference between the standard deviations, $\sigma_{V(S^1+\mathrm{d}S)} - \sigma_{V(S^2-\mathrm{d}S)}$, does not yield a cancellation with certainty.

2.2.3 Constructing Sample Jump-Diffusion Paths

Given that we have discretized time, the approximation to the risk-neutral asset price evolution is given by ([14] p.138)

$$x(t_{i+1}) = x(t_i) + \left(r - \frac{1}{2}\sigma^2\right)(t_{i+1} - t_i) + \sigma \underbrace{[W_{t_{i+1}} - W(t_i)]}_{\sqrt{t_{i+1}-t_i}\ Z} + \underbrace{\sum_{j=N(t_i)+1}^{N(t_{i+1})} \log Y_j}_{P}$$

where $Z \sim \mathcal{N}(0,1)$. Then a procedure for generating a sample path is

1. For each time node t_i from $i = 0$ to N chronologically:
   (a) Generate $Z \sim \mathcal{N}(0,1)$ a standard normal variable;
   (b) Increment the Brownian motion with drift:
   $$x_{BM}(t_{i+1}) = x_{BM}(t_i) + (r - \tfrac{1}{2}\sigma^2)(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i}\ Z\ ;$$
2. Generate jump times $\tau_{j+1} = \tau_j - \log(U)/\lambda$ where $U \sim \mathcal{U}[0,1]$, while $\tau_j < T$;
3. For each j generate $Z_j \sim \mathcal{N}(0,1)$ and then $\log Y_j = \mu + \delta Z_j$;
4. For each i, compute $P = \sum_{j=1}^{\sup_j \tau_j < t_i} \log Y_j$ and set $x(t_i) = P + x_{BM}$;

In the final step, P may be stored as a running total to reduce computation. We create the Brownian motion paths first and then adds the jumps on top, because jumps are additive in x, i.i.d and independent of the Brownian motion. By imposing the rule that a jump must have occurred *before* t_i for $x(t_i)$ to be affected by the jump, we may marginally underestimate the effect of jumps.

We take draws from a standard normal distribution, so we convert $\mathcal{U} \to \mathcal{N}(0,1)$. This is equivalent to describing $W^{-1}(x)$, the inverse normal cumulative distribution function. There is no analytical formula, so we construct an approximation. The Polar-Marsaglia method makes use of points (y_1, y_2) in the unit circle:

1. Draw y_1 and y_2 from $\mathcal{U}[-1,1]$;
2. Compute $R = y_1^2 + y_2^2$;
3. If $R > 1$ (not in unit circle) goto 1.;
4. Compute $\alpha = \sqrt{\frac{2}{R}\log R}$;
5. Set $z_1 = \alpha y_1$ and $z_2 = \alpha y_2$.

Note that $(1-\pi)/4$ of the pairs (y_1, y_2) do not lie in the unit circle, so there is a little wastage. However, an alternative method called Box-Muller requires computation of trigonometric functions, which is more computationally expensive.

To improve the speed of Monte Carlo estimation, we may use two different methods: optimizing the ratio of N to M (the number of time intervals to the number of paths) and antithetic sampling. According to [14], if generating a replication C_i requires a fixed computing time τ, then asymptotically as the total available computation time tends to infinity, we prefer the estimator with the smallest value of $\sigma_i^2\tau_i$. Therefore, for valuing options, we may adjust the number of time intervals N and the number of paths M so that $\sigma_i^2\tau_i$ is minimized.

Antithetic sampling is used to improve the Monte Carlo constant where the payoff is not even. It works best on odd payoffs, but it certainly helps with vanilla options. The idea is that whenever we draw Z from $\mathcal{N}(0,1)$, we also use $-Z$. Hence, we create two Brownian motion paths in x, one a reflection of the other in the drift line. Similarly, for the normal random variable used to generate jump amplitudes.

2.2.4 Results of Monte Carlo Methods

Here we present some numerical results of using Monte Carlo methods. Figure 2.4 compares the analytical jump-amplitude PDF (2.4) e distribution of jumps created through Monte Carlo simulation. With M very large, the two distributions match very well. Notice that the expected jump size occurs at a larger Y than the maximum of the distribution, because of the long tail as $Y \to \infty$. Note that $\delta = 0$ would give a Dirac delta function and $\mu = 0$ would move the maximum to $Y = 0$.

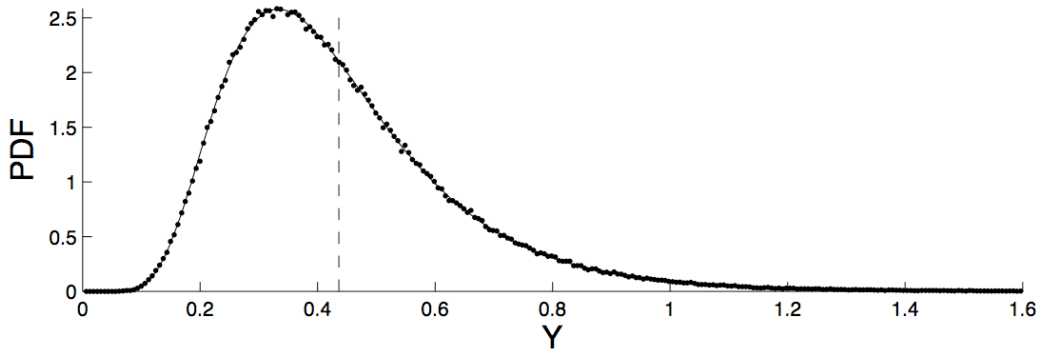


Figure 2.3: The jump amplitude distribution when $g(\cdot)$ is log-normal. The dots represent the PDF from $M = 500,000$ jump simulations using 2^8 bins of a uniform size given by $1.6/2^8 = 0.00625$ in the Y direction. The line is the exact PDF given by (2.4). The dotted line represents the expected jump amplitude, 0.4362.

We plot sample jump-diffusion paths created through simulation on the left of Figure 2.4. The density of the terminal value of the paths is in proportion to the PDF shown on the right of Figure 2.4, but many more sample paths are required to produce the smooth analytical PDF. It is clear that the lower peak of the PDF distribution near $x \approx -0.7$ is because the mean jump amplitude in x is $\mu = -0.92$ and the underlying drift is positive. For $\lambda = 0.2$ they are fewer jumps in the price than for $\lambda = 0.4$ and therefore the respective PDF appears closer to a Gaussian distribution in x. It appears as if all the jumps in the sample paths are negative in x, because of the large negative parameter μ. However, the distribution actually has fatter tails than the normal distribution for both large negative and positive x. If $\mu = 0$, the fattening would be even on both sides and if $\mu > 0$, the fattening would be greater for positive x.

Figure 2.5 shows $\sigma_{MC}/\sqrt{M}$, the convergence of the Monte Carlo estimate to the price of European options as a function of M. If we let one Monte Carlo estimate to the price of an option be P_i, then

$$\sigma_{MC} = \frac{1}{M-1} \sum_{j=1} \left(P_j - \bar{P}\right)^2$$

where $\bar{P}$ is the mean of M Monte Carlo estimates P_i. The fact that the data points in Figure 2.5 asymptote towards the line as $M \to \infty$ means that the convergence is $\mathrm{O}(M^{-1/2})$. The data points are more dispersed for small M, corresponding to the standard deviation of σ_{MC} being greater for small M. In summary, the greater the number of paths, the more accurate the Monte Carlo estimate and the less the variance of this estimate.

Figure 2.6 shows the decrease in the relative error as M increases. We define the relative error as

$$\epsilon_{rel} = \frac{V_{MC} - V_{exact}}{V_{exact}} \tag{2.14}$$

and the data points correspond to the 2-norm of the vector of relative errors divided by $\sqrt{M}$. In this manor, a relative error of 1 for every path where there are M paths corresponds to a data point on the plot at $(M, 1)$. The accuracy certainly does increase with M and there seems to be approximate correlation with the line of convergence $\mathrm{O}(M^{-1/2})$.

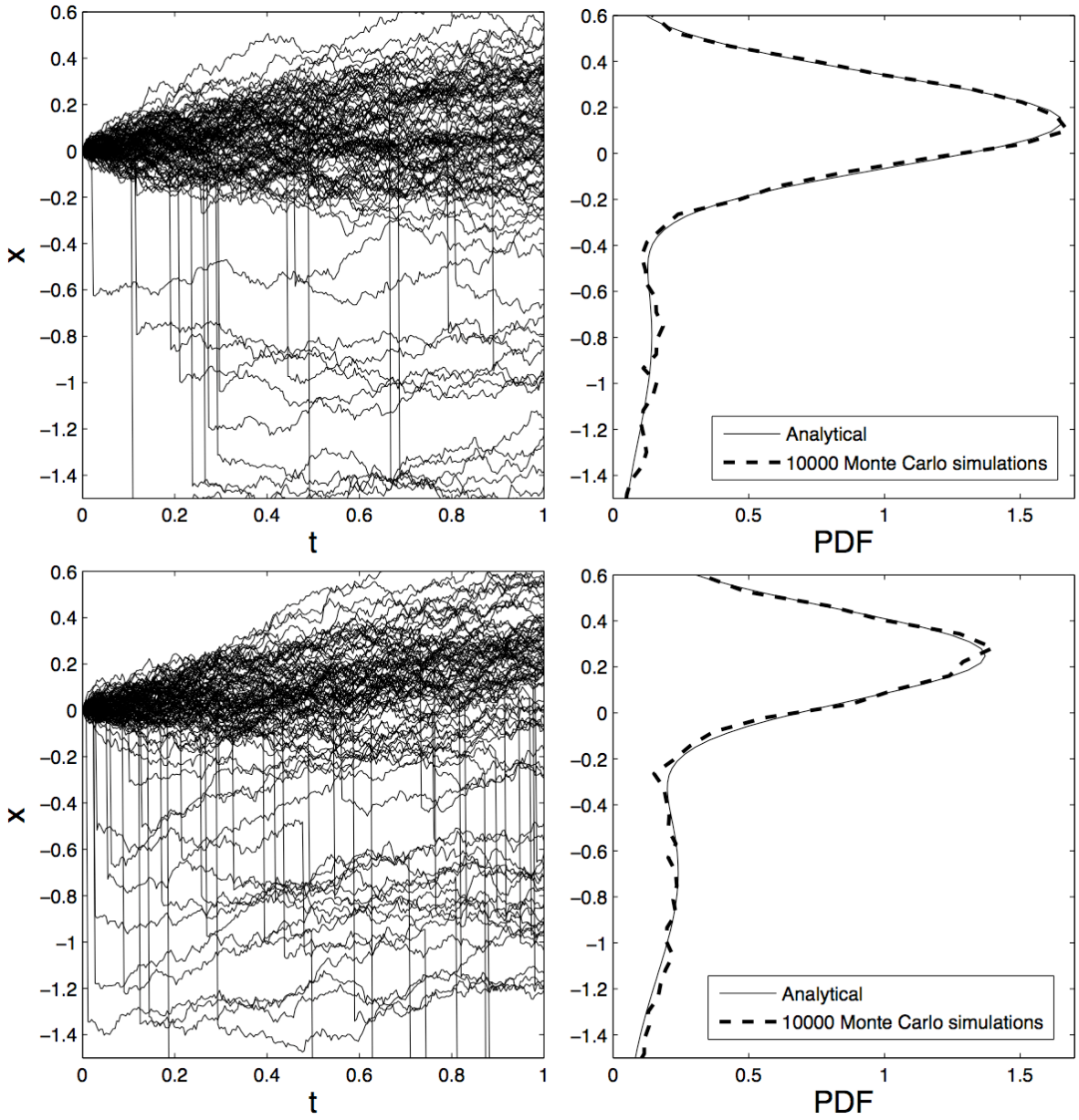


Figure 2.4: 100 simulated paths on the left. The analytical PDF of the terminal asset price (2.5) against the PDF resulting from Monte Carlo simulations on the right. The PDF has been plotted "on its side" so that the x axes are aligned. The top plots both have $\lambda = 0.2$ and the bottom plots both have $\lambda = 0.4$. There are $N = 2^8$ time intervals, $M = 10,000$ and the maturity is $T = 1$. An unrealistically large value of λ is chosen so that the peak in the PDF around $x \approx -0.7$ due to jumps is clearly visible. The Monte Carlo PDF is created using 2^7 bins of uniform size in x, each equal to $2.1/2^7 \approx 0.0164$

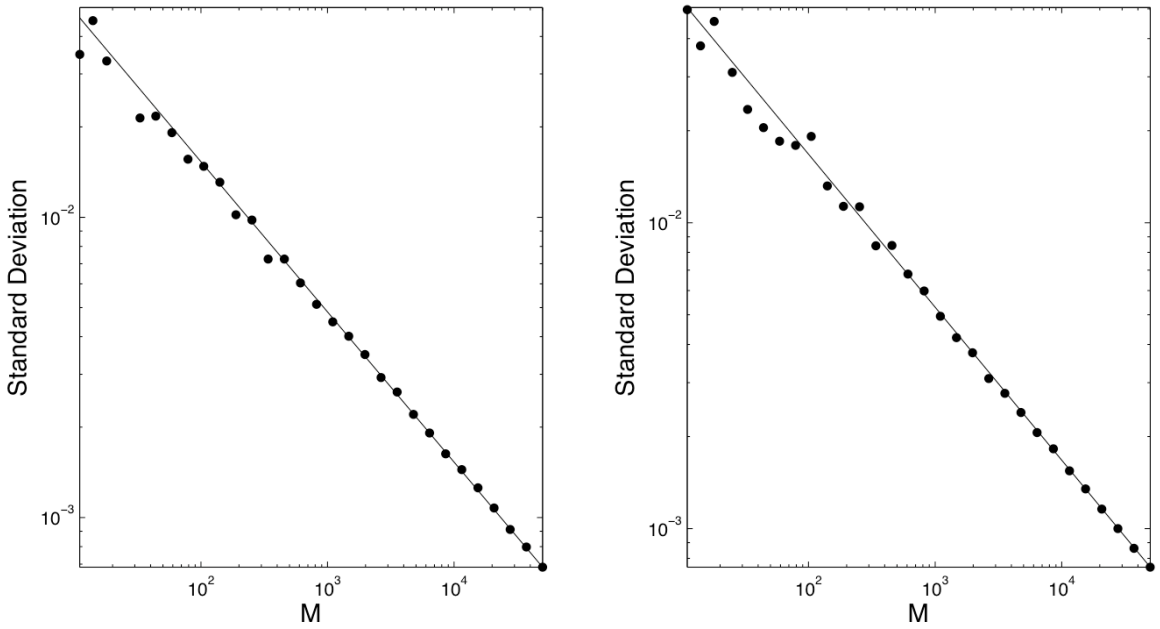


Figure 2.5: Convergence of the Monte Carlo estimate for the value of a European put option on the left and call option on the right. Each data point corresponds to the estimated standard deviation of the mean of M paths. The line represents the convergence rate $1/\sqrt{M}$. There are $N = 2^8$ time intervals and $T = 1$.

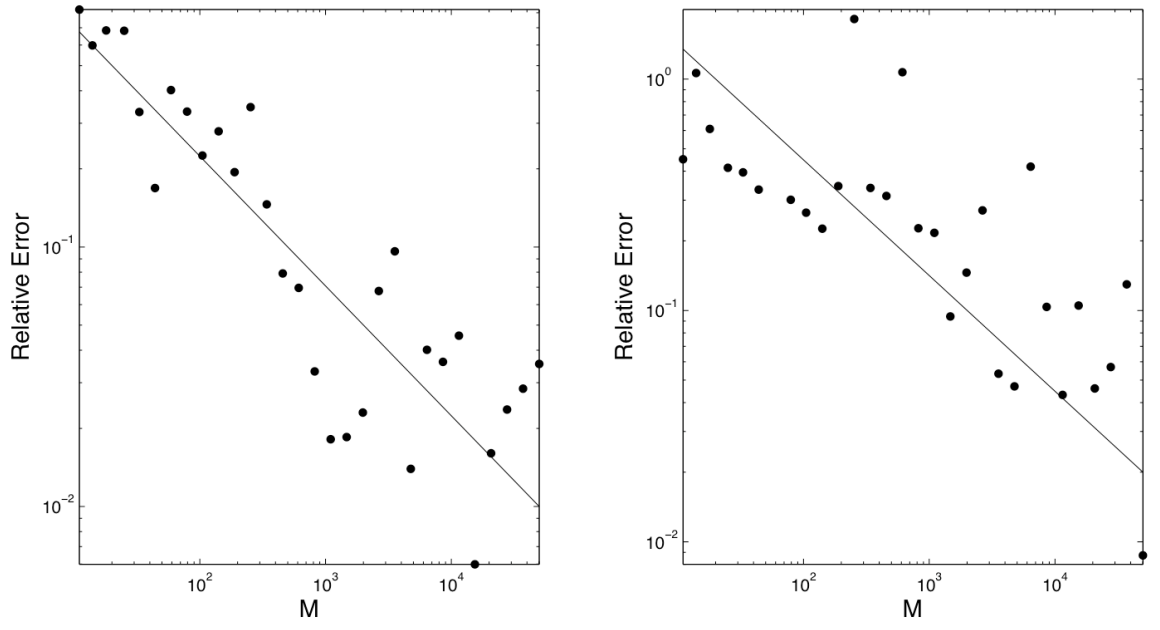


Figure 2.6: Convergence of the Monte Carlo estimate for the value of a European put option on the left and call option on the right. Each data point corresponds to the 2-norm of the relative error of M paths divided by $\sqrt{M}$. The line represents the convergence rate $1/\sqrt{M}$. There are $N = 2^8$ time intervals and $T = 1$.

2.3 Numerically Solving the PIDE

2.3.1 Finite Difference Approximation to the PIDE

In this section, we formulate a numerical method to approximately solve the PIDE (2.9). For large $|x|$, the option price behaves asymptotically like the payoff function. Therefore, given a domain, $x \in [-\alpha, \alpha]$, we set $u(\tau, x) = H(e^x)\ \forall\ x \in \mathbb{R} \setminus [-A, A]$.

We discretize (2.9) using a finite difference scheme. We must truncate the integral over jump-amplitudes so that we can approximate it: let $\log Y \in [B_l, B_r]$. Using the notation of [7] (p.414), the right hand side of (2.9) may be expressed as

$$\mathcal{L}u = \frac{\sigma^2}{2}\frac{\partial^2}{\partial x^2} + \left(r - \frac{\sigma^2}{2} - \alpha\right)\frac{\partial u}{\partial x} + \lambda \int_{B_l}^{B_r} g(y)u(\tau, x+y)\mathrm{d}y - \lambda u, \qquad (2.15)$$

$$\text{where} \quad \alpha = \lambda \int_{B_l}^{B_r} (e^y - 1)g(y)\mathrm{d}y.$$

We may split $\mathcal{L}$ into two operators: $\mathcal{L} = \mathcal{D} + \mathcal{J}$, where $\mathcal{D}$ represents the diffusive part and $\mathcal{J}$ the jumps. This is a generalization of implicit-explicit methods [5], but there are alternatives, such as spitting-predictor-corrector methods [12] (p.189). To approximate the partial differentials, we apply standard finite differences.

We use a central difference for the drift term, even though this may lead to oscillations in the solution. This is because [24] (p.22) states that under reasonable parameter values, there are only very slight discrepencies in the accuracy of central differences in comparison to forward or backward differences. In addition, central differences give second order accuracy in x. Given the approximation u_i^0 to $u(\tau_n, x_i)$, the numerical schemes for the operators $\mathcal{D}$ and $\mathcal{J}$ are

$$(\mathcal{D}u^n)_i = \frac{\sigma^2}{2}\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} + \left(r - \frac{\sigma^2}{2} - \hat{\alpha}\right)\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x},$$

$$(\mathcal{J}u^n)_i = \lambda \sum_{j=-K}^{K} g_j u_{i+j}^n - \lambda u_i^n, \qquad (2.16)$$

with $n \in \{0, ..., N_x\}$ and $\Delta x = 2A/N_x$. The approximation to the drift from jumps is given by

$$\hat{\alpha} = \lambda \sum_{j=K_l}^{K_r} (e^{y_j} - 1)g_j \qquad (2.17)$$

and the cumulative PDF for jumps is approximated by

$$g_j = \int_{(j-1/2)\Delta x}^{(j+1/2)\Delta x} g(y)\mathrm{d}y.$$

Note that if $g(\cdot)$ is a Gaussian distribution, the cumulative PDF is not available in closed form and this integral must be approximated, e.g. using a quadrature rule. For the region outside the domain, we impose the payoff $u_i^n = H(x_i)$ for $i \notin \{0, ..., N_x\}$.

Let us consider a (θ_D, θ_J) scheme, where θ_D represents the implicit/explicit bias for the differential operator and θ_J the implicit/explicit bias for the integral operator. In order to numerically solve the PIDE, at each time level we need to solve the system

$$\big[I - (1-\theta_D)\Delta\tau D - (1-\theta_J)\Delta\tau J\big]u^{n+1} = [I + \theta_D\Delta\tau D + \theta_J\Delta\tau J]u^n. \quad (2.18)$$

Here, I is the identity matrix, D and J are matrices representing the discretizations of the differential and integral operators, with $N_x + 1$ rows and columns. u^n is a vector of length $N_x + 1$ representing the approximate solution at time level n. We have $\theta_D, \theta_J \in [0, 1]$, where 0 is implicit and 1 explicit. [5] claims that $\theta_D = 0$, $\theta_J = 1$ and using Runge-Kutta methods to solve the integral term results in a stable scheme.

2.3.2 Penalty Method for American Options

In order to price American options, we may solve the linear complementarity problem

$$V_\tau - \mathcal{L}V \geq 0 \quad (2.19)$$

$$V - V^* \geq 0 \quad (2.20)$$

where one of the equations holds with equality. $\mathcal{L}$ is defined by (2.15) and V^* is the payoff function for the option. [10] explains how a penalty method may be used to enforce the equality of (2.20) in the exercise region and equality of (2.19) otherwise. To do so, we replace the linear complementarity problem by the penalty problem

$$V_\tau = \mathcal{L}V + \rho \max(V^* - V, 0)$$

where ρ is much larger than the order of the other terms in the equation. This can be approximated using the finite difference scheme proposed in section 2.3.1:

$$\begin{aligned}\big[I - (1-\theta_D)\Delta\tau D + \Delta\tau P(V^{n+1}) - (1-\theta_J)\Delta\tau J\big]V^{n+1} \\ = \big[I + \theta_D\Delta\tau D + \theta_J\Delta\tau J\big]V^n \\ + \big[\Delta\tau P(V^{n+1})\big]V^*\end{aligned}$$

Let ρ = 1/tolerance (typically $O(10^6)$) so that the nonzero entries of the diagonal matrix P dominate the scheme, enforcing equality of (2.20), where P is given by

$$\text{where} \quad P(V^n)_{ii} = \begin{cases} \rho, \text{ if } V_i^n < V_i^* \\ 0, \quad \text{O/W.} \end{cases}$$

If $\theta_D = 0$, the treatment of the differential operator is implicit and if $\theta_J = 0$, the treatment of the integral operator should also be implicit. However, solving the system $JV^{n+1} = b$ where b is a vector is computationally expensive, because J is dense. Since D is tridiagonal, an iterative algorithm can be used, where each matrix solve is only for D. Note that this does not strictly treat the integral operator implicitly. An iterative method is (as described in [10], p.10):

Let $V^n = (V^{n+1})^0$
Let $\hat{V}^k = (V^{n+1})^k$
Let $\hat{P}^k = P\big((V^{n+1})^k\big)$
$k \leftarrow -1$
do
$k \leftarrow k + 1$
Solve the tridiagonal system:

$$\big[I - (1-\theta_D)\Delta\tau D + \Delta\tau\hat{P}^k\big]\hat{V}^{k+1} = [I + \theta_D\Delta\tau D + \theta_J\Delta\tau J]V^n + \Delta\tau\hat{P}^k V^* + (1-\theta_J)\Delta\tau J\hat{V}^k$$

while $\max_i \left(\dfrac{\hat{V}_i^{k+1} - \hat{V}_i^k|}{\max\left(1, |\hat{V}_i^{k+1}|\right)} \right) \geq$ tolerance

Rather than using an iterative method with (possibly many) matrix multiplies, [10] (p.6) points out that the integral evaluation may be performed in $O(N_x \log N_x)$ using fast Fourier transforms (FFTs) at each time level. The FFTs are the major contributor to the computational expense of this method, so it would certainly be quicker. However, for the accuracy that we require, $N_x \sim 10^3$ in general, giving a manageable $O(10^6)$ operations at each time-level for the dense matrix solve. To further improve the speed, dynamic time-stepping may be used, as explained in Appendix A.1.2.

2.3.3 Numerical Results for the PIDE

Here we present the results of the numerical solution to the PIDE. To obtain a satisfactory level of accuracy, the domain of x is $[-6, 6]$ for most of the following numerical experiments. We use $\theta_D = \theta_J = \frac{1}{2}$, the Crank Nicolson scheme. The overall aim of solving the PIDE with a penalty term is to price American options, but the value of an American option under jump-diffusion may not be corroborated, since there is no analytical formula. However, as shown in Figure 2.7, the price asymptotes to the Merton price of a European option as $S \to \infty$ and to the Black-Scholes price of an American option as $S \to 0$, which is intuitively sound.

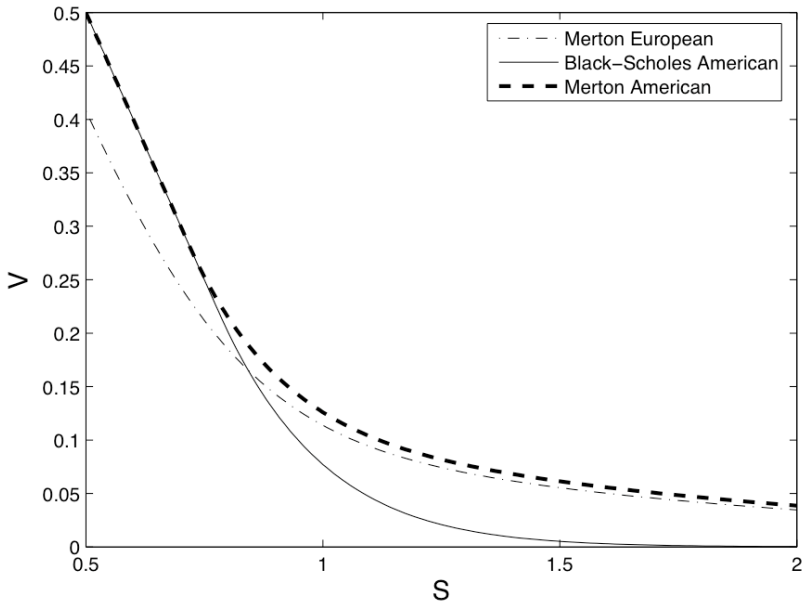


Figure 2.7: The value of a put option. This compares the Merton American value generated through numerical solution of the PIDE with two other option values. There are $N_x = 1,500$ spatial intervals, $N = 30$ time intervals, with maturity $T = 2$.

Figure 2.8 displays the exercise boundary over time for the American put option under Merton's jump diffusion, with the exercise region below each curve. The greater the jump intensity the smaller the exercise region, because a large downard jump may occur. Note that in the Black-Scholes case, American options are always exercised *on* the boundary, whilst under Merton's jump-diffusion, a jump may have move the price *past* the boundary into the interior of the exercise region.

Figure 2.9 illustrates that the prices follow quadratic convergence in N_x, as we hoped. We cannot actually test the case of an American put option under Merton's jump-diffusion, but we can safely extrapolate from these errors and say that it follows quadratic convergence. It is important that the ratio of N to N_x always remains fixed to ensure that this convergence rate is maintained.

Figure 2.10 compares the PIDE solution to the known price of call and put options under Black-Scholes ($\lambda = 0$) and jump-diffusion through the relative error, as defined by (2.14). For the put option, as $S \to \infty$, the relative error seems to increase substantially, but this is because $V_{exact} \to 0$. Similarly, for the call option as $S \to 0$. Notice that the relative error in the Merton European price is smaller in each of these cases, because as $V_{exact} \to 0$ the Merton price lies above the Black-Scholes price. The strike is at $K = 1$, but the solutions do not have a sharp change in accuracy in its vicinity, which is advantageous.

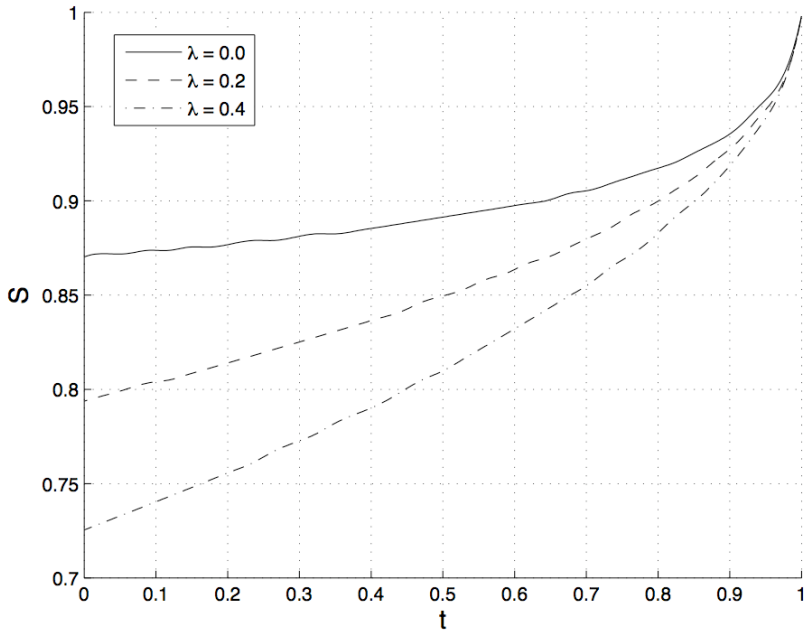


Figure 2.8: The exercise boundary of a Merton American put option through numerical solution of the PIDE for several values of λ. $N_x = 2,000$, $N = 34$, $T = 1$.

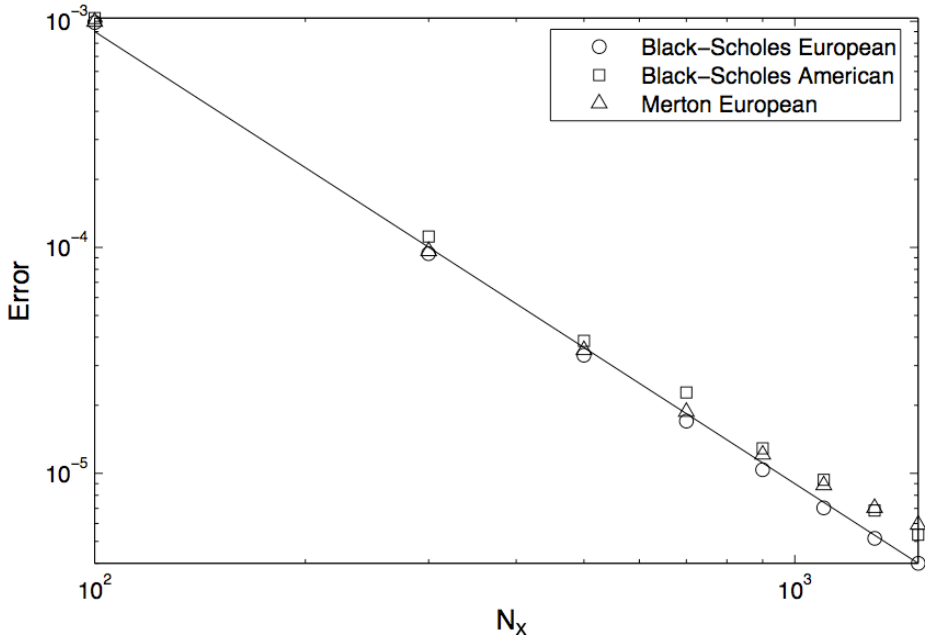


Figure 2.9: Convergence of the value of a put option obtained from numerical solution of the PIDE, where the ratio $\Delta\tau = 2\Delta x$ is held constant throughout. The absolute error is plotted: $||V_{\text{exact}} - V_{\text{PIDE approx}}||_2$. The line represents quadratic convergence: $1/N_x^2$. The maturity is $T = 0.25$.

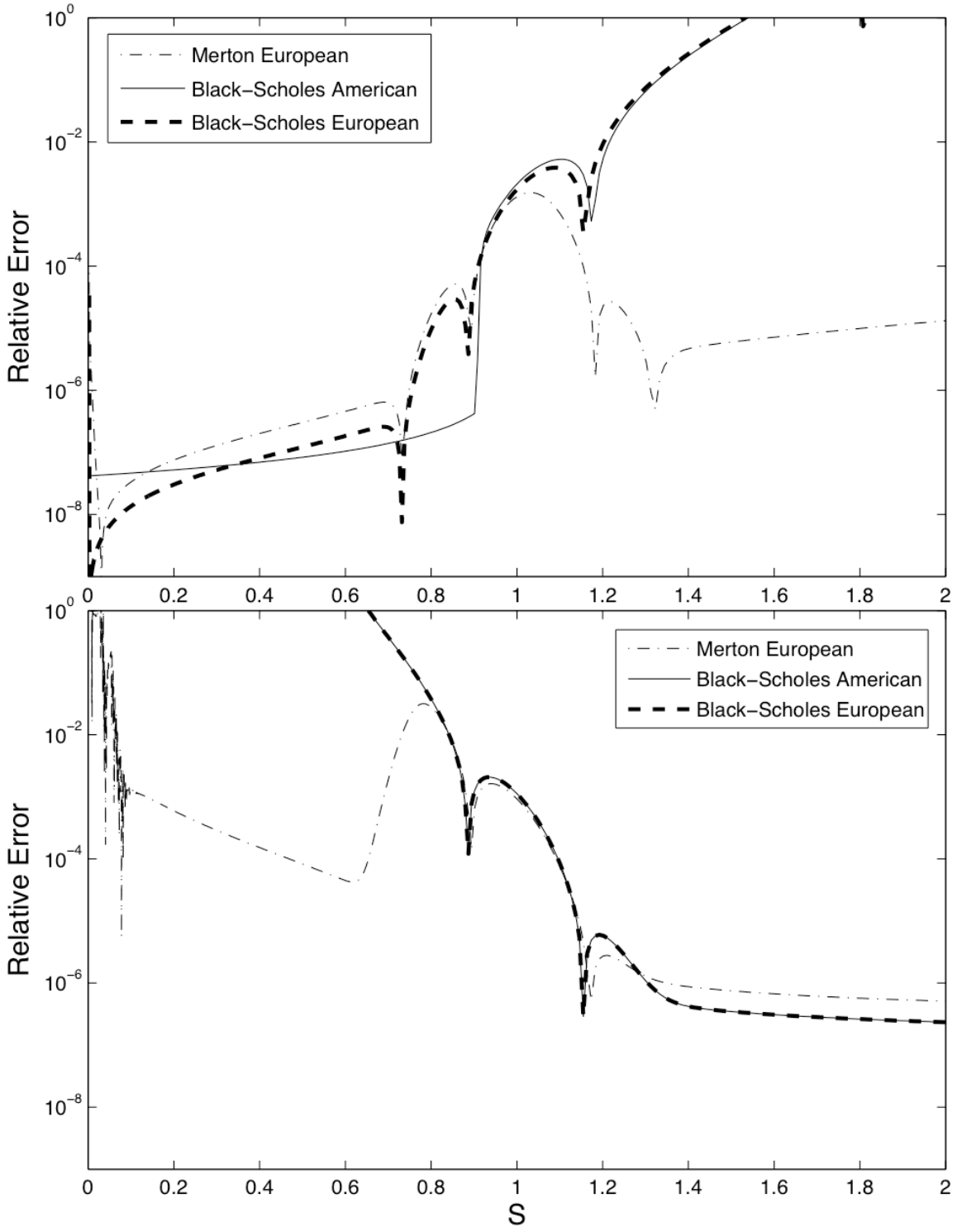


Figure 2.10: The absolute size of the relative error in the value of a put option (top) and call option (bottom). This compares the analytical formulae to the numerical solution of the PIDE with $Nx = 1,500$ and $Nt = 30$. Owing to the nonexistence of an analytical formula for the American put price, the Black-Scholes American "exact" price used to compute the relative error is actually a very accurate solution obtained through a different numerical technique, kindly provided by Alex Prideaux [29].

Chapter 3

Hedging Strategies

3.1 Hedging in an Incomplete Market

3.1.1 Delta Hedging under Black-Scholes

As observed in [14] (p.19), in a complete market, where an option can be perfectly replicated by holding a combination of other instruments, the price of the option is equal to the cost of the exact hedge. Hence, the price of the hedging instruments uniquely determines the price of the target option. Conversely, in an incomplete market perfect replication through other instruments is impossible and hence the price of an option cannot be uniquely determined by other market prices.

In the Black-Scholes model, the market is complete: a target option may be perfectly replicated by following a delta-hedging strategy where the quantity of asset held Δ is continuously updated so that

$$\Delta(t) = \frac{\partial V}{\partial S}(S, t) \tag{3.1}$$

at all times, which is the target option's delta. The rest of the proceeds from the sale of the option are invested at the risk-free rate in b bonds, each with value $B(0)$ at time 0. Provided trading in the underlying asset may be performed in infinitesimally small time increments at all times $t \in [0, T]$, where the target option matures at time T, the portfolio's value is

$$\Pi(S, t) = \Delta(t)S(t) + b(t)B(t) - V(S, t) \tag{3.2}$$

which is zero and hence the option is perfectly hedged. A proof is in Appendix A.2.1.

In reality, an infinite number of trades in the underlying cannot be performed and hence trading at (possibly variable) discrete time intervals occurs. Therefore, even for delta hedging under the Black-Scholes model, there is hedging error.

To evaluate the performance of a particular hedging strategy we use the metric (as suggested in [17])

$$\text{Relative P\&L} = e^{-rT}\frac{\Pi(S_T, T)}{V(S_0, 0)}, \tag{3.3}$$

the relative profit and loss (P&L). Using Monte Carlo simulation we can estimate the PDF of this quantity given a hedging strategy.

Figure 3.1 tests delta hedging in a Black-Scholes world. The plots show the distribution of relative P&L (3.3) resulting from Monte Carlo simulation of delta hedging. Each curve corresponds to a different, fixed number of rebalance times, with uniform rebalance intervals. As $N \to \infty$, the distribution appears to tend towards a Dirac delta function at the origin. This is desirable, since it means that the hedging error is decreasing in $N \; \forall \; N$.

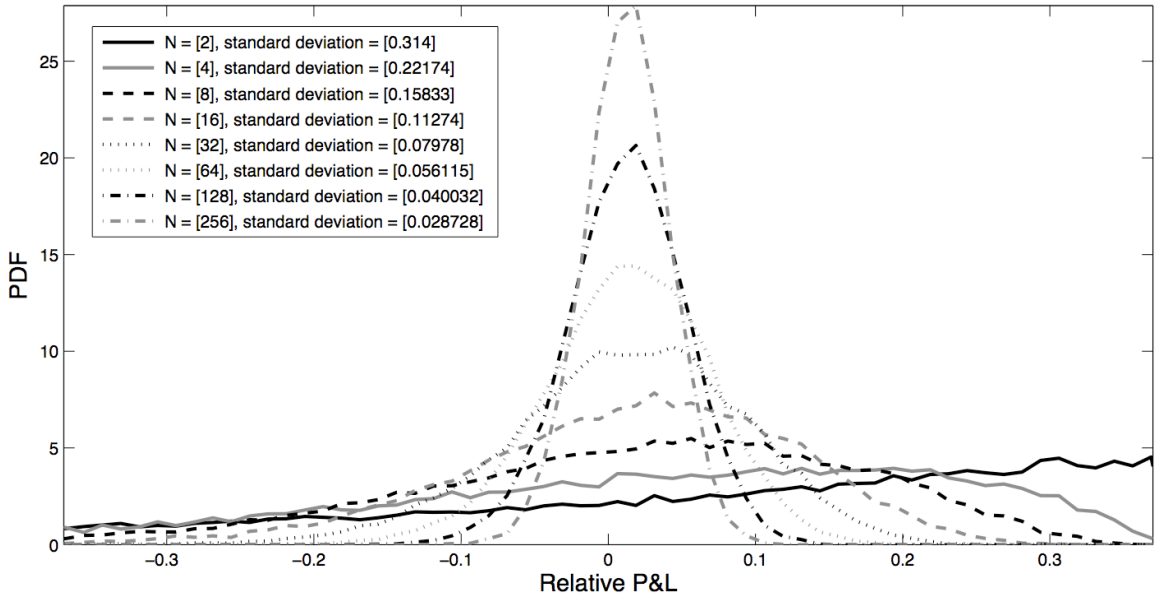


Figure 3.1: The relative P&L resulting from delta hedging under the Black-Scholes model with r and σ as in Table 2.1. Each curve is the distribution for a specific number of rebalance intervals N. There are $M = 20,000$ asset path simulations and option is a European put with maturity is $T = 0.25$. The standard deviation is measured from all the data, including points outside the domain of the plot.

The mean of each distribution is very close to zero, but the standard deviation is much larger. The standard deviation $\sim 1/\sqrt{N}$ where N is the number of rebalances. For example, reading off Figure 3.1, $0.314/\sqrt{(256/2)} = 0.0278$ (to 3 significant figures), which is almost 0.028728. Therefore, delta hedging is succesful under the Black-Scholes model, provided the payoff is continuous. Next, we investigate delta hedging when there are jumps in asset price.

3.1.2 Delta Hedging under Jump-Diffusion

In the case of hedging options in a world with jumps, delta hedging is not optimal, unless the jumps are diversifiable. Merton argues in [26] that if jumps only affect a limited number of assets simultaneously, holding a well-balanced portfolio cancels out the jump risk. [6] suggests hedging with a Black-Scholes transition PDF that is similar to that of Merton's jump-diffusion (but not the same). The compensated variance $\bar{\sigma} = \sqrt{\sigma^2 + \lambda(\mu^2 + \delta^2)}$ is used instead of σ to at least partially take into account the effect of jumps. However, it is better to follow Merton's hedging strategy: delta hedge using Merton's jump-diffusion delta.

If jumps affect all the world's markets[1], then the jumps are not diversifiable. In this case, delta hedging is not successful, as Figure 3.2 demonstrates. The mean relative P&L appears positive, but in fact the mean is almost zero for every choice of N. Therefore, a lot of the weight is in the left hand tail of the distribution.

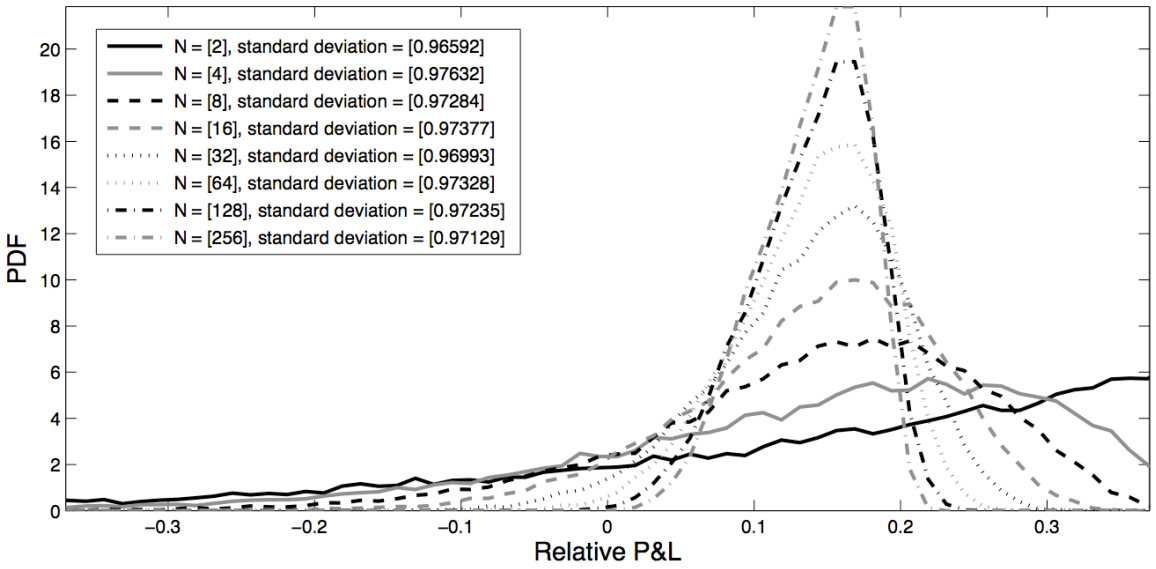


Figure 3.2: Equivalent to Figure 3.1, except the asset price evolution is governed by Merton's jump-diffusion with parameters as in Table 2.1.

In Figure 3.2, the spread of the distributions appears to decrease with increasing N. However, the standard deviation is approximately the same, for all N, the skew is negative and the kurtosis remains large. This is in sharp contrast to Figure 3.1 where the standard deviation systematically fell with increasing N. Therefore, delta hedging in Merton's jump-diffusion world has a material probability of failing. When the asset price jumps, large losses are incurred.

[1]See section 1.1 for some examples of worldwide crashes.

3.1.3 Hedging Jumps using Options

If each jump has one of a finite number N_J of possible amplitudes, then a target option may be perfectly hedged using N_J different options, the underlying and bonds. The hedging instruments span the space of possible target option values given a movement in the underlying S. Let the target mature at time T and the hedging options mature at time u ($u < T$).

Let there be one jump amplitude Y, one short-dated hedging instrument with value I, let $S = 1$ and suppose we study a small period of time $[t_0, t_0 + \mathrm{d}t]$. Assume further, that the price of the hedging instrument, the price of the target option, a constant payoff (bonds) and S (underlying) are all linearly independent as functions of S. At a time t with asset price S, the portfolio value is $\Pi(S, t)$ given by

$$\Pi(S, t) = \Delta(t)S(t) + b(t)B(t) + \phi(t)I(t) - V(t)$$

At $t = t_0$, $S = S_0$ there exists a linear combination $\Pi(S_0, 0)$ such that

$$\begin{aligned} \Pi(S_0, t_0) &= 0 \\ \frac{\partial \Pi}{\partial S}(S_0, t_0) &= 0 \\ \Pi(Y S_0, t_0 + \mathrm{d}t) &= 0 \end{aligned}$$

As $\Delta t \to 0$, there will be at most one jump and the delta hedging condition neutralizes the risk posed by Brownian motion. Therefore, the hedge is perfect. For each additional possible jump amplitude, one more independent hedging instrument must be added, as proposed in [2] (p.22).

If there are an infinite number of possible jump amplitudes, perfect replication requires an infinite number of different hedging instruments (such as one type of option at a continuum of strikes). However, in reality these are not available. There are an infinite number of possible jump amplitudes under Merton's jump-diffusion, because each jump is log-normally distributed. Therefore, a perfect hedge does not exist and the market is incomplete.

In Merton's world, the best we can hope to do is minimize the risk of the hedge, the value of $\Pi(Y S_0, t_0 + \mathrm{d}t)$, where $Y \in \mathbb{R}$. It is impossible to have $\Pi(Y S_0, t_0 + \mathrm{d}t) = 0$ $\forall\, Y \in \mathbb{R}$ (except in trivial cases, such as $V = S$). Instead, we choose a weighting for the world and approximately minimize the risk using a finite number of short-dated hedging options.

3.1.4 Semi-static and Dynamic Hedging

To hedge a target option, we follow a set of rules throughout its life, the hedging strategy. This dictates when we alter our hedge, the instruments we employ and the weights of each. If we have access to hedging options with a continuum of strikes, we may perfectly replicate any future option value $V(S,t)\ \forall\ S \in \mathbb{R}$ through a linear combination of the hedging instruments. This is a pure linear algebra problem. However, in the market we only have a limited set of strikes at our disposal. Therefore, the hedging strategies we study have the aim of replicating the target option's value as closely as possible at some future time.

Imagine that we have written a European option that we would like to hedge. We have at our disposal the underlying, bonds and possibly several other options. Since the target option is European, the value of the portfolio at every infinitesimal moment $t \in (0,T)$ is irrelevant. If the shortest-term hedging option expires at some $u \in (0,T)$ then it is the value of the hedge at $t = u$ that is important (see Figure 3.3 for the timeline). Hence we would like to create a hedge at $t = 0$ that does the best possible job of matching the target option's value at time u. Therefore, we minimize the risk in the long term. This is semi-static hedging.

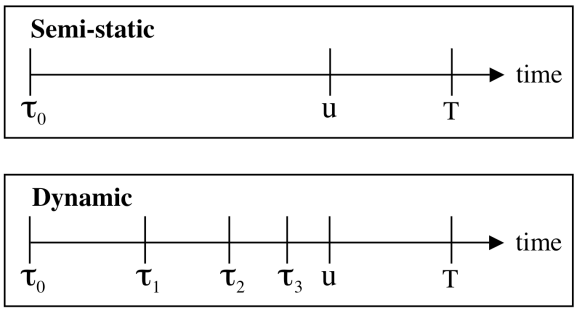


Figure 3.3: A timeline of hedge rebalances under semi-static and dynamic hedging.

If we have written an American option, the holder may exercise it at any time during its life. Therefore, we want to match the value of the option $\forall\ \tau_i \in [0,T]$ (see Figure 3.3 for the timeline). Using dynamic hedging, we create an initial hedge at τ_0 and rebalance it at times τ_i, $i \in \{1,2,...\}$. At time τ_i we hedge to match the target option's value at time τ_{i+1}. At each rebalance we change the hedge to reflect the change in S. Therefore, dynamic hedging may be viewed as local risk minimization. We assume the available hedging options are American and mature at time u. If one of these becomes exercisable, we exercise it and replace it with a new option, so that we always have the same number of available hedging instruments.

3.1.5 Weighting Function and Approximation Method

The hedging instruments span a finite-dimensional space in which the target option does not lie. Therefore, we project the target option onto the space. Let the target option mature at T and hedging options mature at $u < T$. As we shall see, a least squares approach results in an orthogonal projection. Figure 3.4 shows a possible replication strategy. However many hedging instruments we use, the target and hedging curves will never match exactly, but even a few well chosen instruments can do a very good job. In Figure 3.4, the replication time is u and therefore the hedging options' values are their payoffs. The value of a linear combination of hedging options, the underlying and bonds is piecewise linear in S and non-smooth at the strikes.

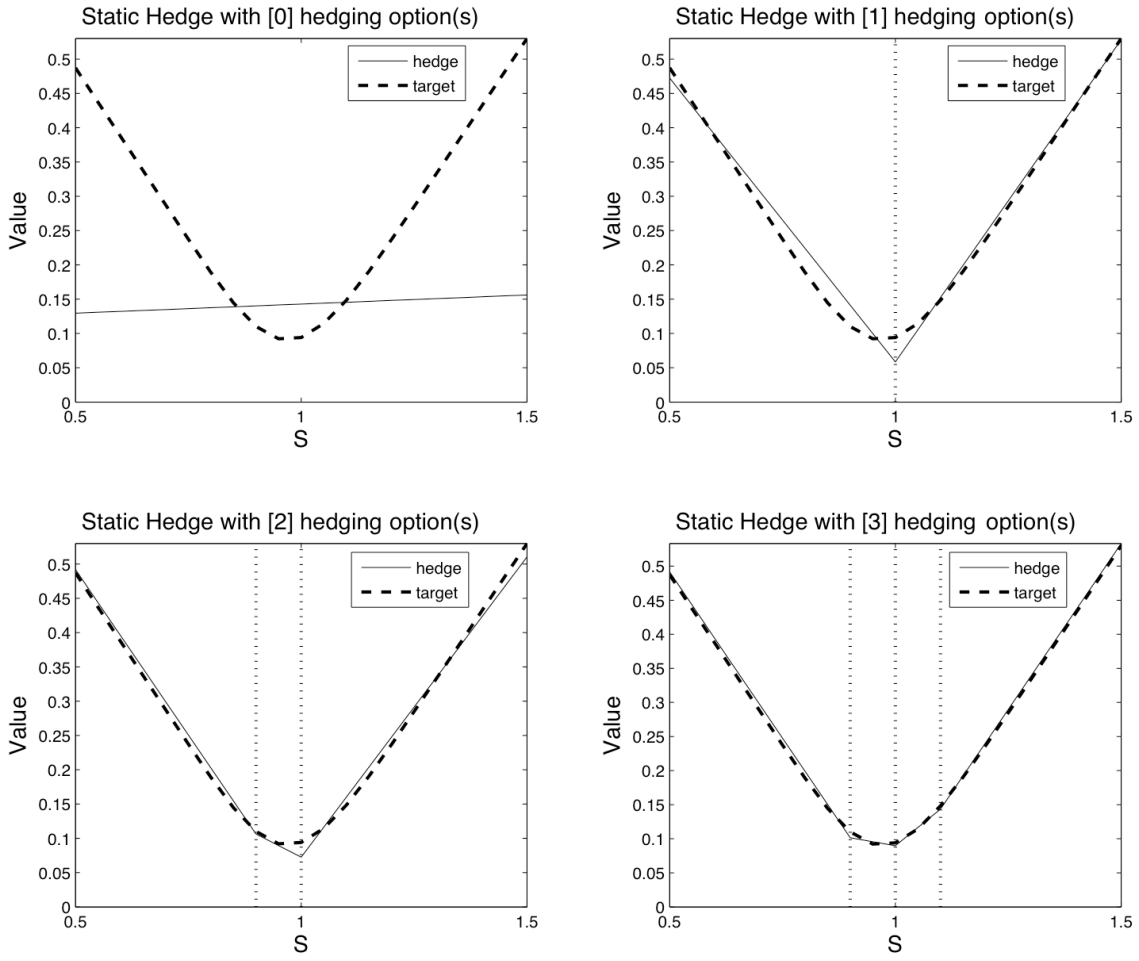


Figure 3.4: Examples of approximating the target option's value at a future time with a hedge of options, the asset and bonds. All options are European, the target option is a straddle maturing at $T = 0.5$, the hedging options are puts maturing at $u = 0.25$, the time at which the hedge aims to replicate the target. The vertical dotted lines show the strikes of the hedging options.

There is always nonzero hedging error, the vertical difference between the two curves in Figure 3.4. We must decide for which values of S the error should be minimal. If we expect the underlying asset to be near the vicinity of $\bar{S}$ at time u, we may choose to make the hedging error very small for a range of S near $\bar{S}$. We express the importance of the domain S through a weighting function $W(S)$ where $\int_0^\infty W(S)\mathrm{d}S = 1$ and $W(S) \geq 0 \; \forall \; S$, as in Figure 3.5.

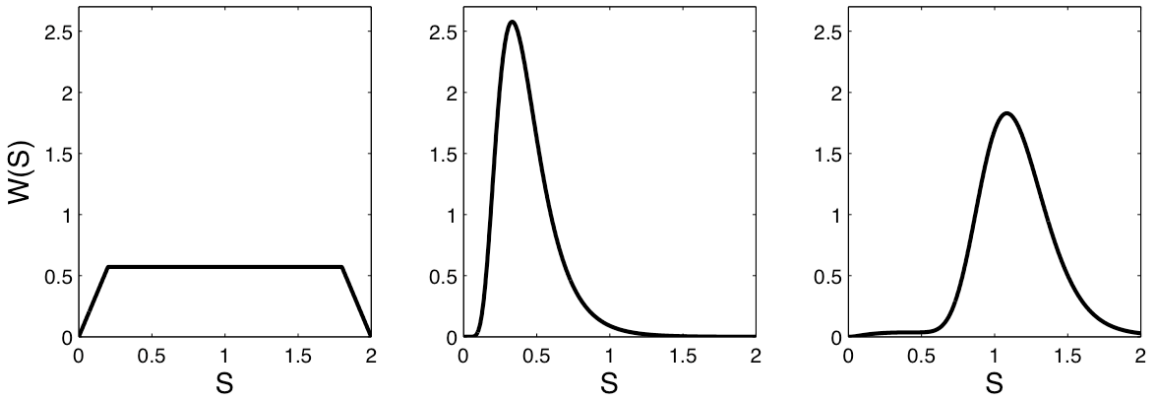


Figure 3.5: Some possible weighting functions. The leftmost is a uniform-like function [17] where the expected distribution is unknown. In the centre is the jump amplitude distribution (2.4). The rightmost is the transition PDF (2.5) for $T = 1$.

Once we have expressed our weighting preference $W(S)$, we choose a method of approximating the target option's value to obtain the strikes and weights of the hedging options. However, in the marketplace there are a limited number of strikes, so we let the hedging strikes be $\{0.50S_0, 0.55S_0, ..., 1.50S_0\}$. The hedging world available to us may be thought of as points in the 3D space depicted in Figure 3.6. In the limit of no constraint on all axes we may perfectly hedge, even in an incomplete market. Transaction costs justify the constraints on our hedging world: less liquid strikes, a larger number of instruments and more frequent rebalancing all heighten costs.

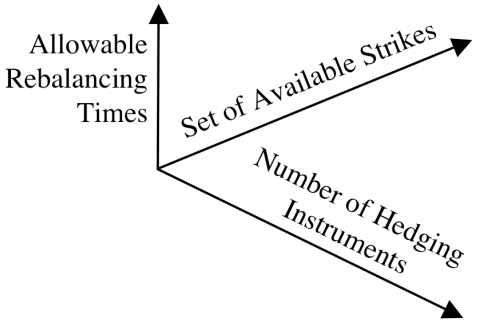


Figure 3.6: The 3 main hedging constraints (arrows represent increasing freedom).

3.2 Formulating Strategies

3.2.1 Hedging with Weighting Proportional to Γ

One method of semi-static hedging of European options is proposed in [6]. Given a target option maturing at T and shorter term hedging options maturing at $u < T$, the weights of the hedging options are proportional to the target's Γ at $t = u$. This does not change as S evolves and hence the hedge remains static over $[0, u]$. Furthermore, an accurate description of the underlying price dynamics is only required over $[u, T]$.

However, a perfect static hedge uses an infinite number of hedging instruments (a continuum of strikes), which is impossible in the market. We must approximate the strategy through discretizing in the strike direction. As mentioned in [6] (p.3), this is analogous to the discretization of time required when we dynamically hedge. We should choose the number of hedging options so that we balance the hedging error and the cost of transacting in these options[2].

As mathematically justified in Appendix A.2.2, the European option price is

$$V(S,t;K,T) = \int_0^\infty w(\mathcal{K})V(S,t;K,u)\mathrm{d}\mathcal{K}$$
$$\text{where} \qquad w(\mathcal{K}) = \frac{\partial^2 V}{\partial \mathcal{K}^2}(\mathcal{K},u;K,T). \tag{3.4}$$

We approximate this integral discretely using $\zeta \in \mathbb{N}$ hedging options:

$$\int_0^\infty w(\mathcal{K})V(S,t;K,u)\mathrm{d}\mathcal{K} \approx \sum_{j=1}^{\zeta} \mathcal{W}_j V(S,t;\mathcal{K}_j,u).$$

Now we need to choose the strikes $\mathcal{K}_j$ and the weights W_j using a rule to approximate the integral. Ideally, the approximation converges to the exact functional as $\zeta \to \infty$.

As in [6], we determine strikes and weights $\{\mathcal{K}_j, \mathcal{W}_j\}_{j=1}^{\zeta}$ through the Gauss-Hermite quadrature rule. The reason we do so is that the transition PDF in $x = \log S$ under Merton's jump-diffusion (2.5) is a weighted sum of Gaussian distributions. In Appendix A.2.3 we detail the Gauss-Hermite quadrature rule, its nodes and weights.

We need a map to convert Gauss-Hermite nodes and weights $\{x_j, w_j\}_{j=1}^{\zeta}$ to strikes and option weights $\{\mathcal{K}_j, \mathcal{W}_j\}_{j=1}^{\zeta}$. In [6], the Black-Scholes analytic formula for Γ motivates the choice of map for the strikes,

$$\mathcal{K}_j = K \exp\left\{x_j \bar{\sigma}\sqrt{2(T-u)} - (r + \bar{\sigma}^2/2)(T-u)\right\}.$$

[2]Transaction costs are discussed briefly in the Conclusion of this paper.

This map takes into account the expected drift and diffusion of the asset price through time [u,T]. Due to the lack of strikes available in the market, we must choose available strikes nearest $\mathcal{K}_j$. As in [6], we do not alter the weights to compensate for this mis-striking, but it would be better to do so. The option weights are

$$\mathcal{W}_j = \frac{w(\mathcal{K}_j)\mathcal{K}_j\bar{\sigma}\sqrt{2(T-u)}}{e^{-x_j^2}}w_j$$

where $\bar{\sigma}^2 = \sigma^2 + \lambda(\mu^2 + \delta^2)$ is the compensated variance and $w(\cdot)$ is given by (3.4).

Recall that the option price under jump-diffusion is a weighted sum of Black-Scholes option prices. Therefore, it would be best to use a separate Gauss-Hermite hedge for each Black-Scholes option component, rather than approximating with the single "closest" Black-Scholes PDF. However, this would use a large number of strikes.

3.2.2 Least Squares Hedging

Here we choose to minimize the change in portfolio value squared, at a certain time in the future, weighted to reflect our expectation of asset price movement. This is relevant to both semi-static and dynamic hedging. Mathematically, the problem is

$$\min_{\boldsymbol{\phi}} \int_0^\infty [\Delta_J \Pi]^2 W(Y) \mathrm{d}Y \tag{3.5}$$

where Π is the portfolio value and $W(\cdot)$ is the weighting function. The change in the portfolio value due to the asset price moving from S to SY over our chosen time interval for the hedge is given by $\Delta_J \Pi = \Pi(YS) - \Pi(S)$. The portfolio is

$$\Pi(S,t) = -V(S,t) + \boldsymbol{\phi}(t).\mathbf{I}(S,t) + b(t)B(t)$$

where $\boldsymbol{\phi}$ is a vector representing the weight of each of the hedging instruments, whose values are given by the vector $\mathbf{I}$. The quantity of bonds is b, each with value B.

The problem (3.5) results in an orthogonal projection under the L_2 norm from the target option value onto the space spanned by the hedging option values, the underling and bonds, as mentioned in [7]. Two common choices for the weighting function are the transition PDF and the jump size PDF. The former is conventionally used for static hedging and the latter for dynamic hedging.

In Appendix A.2.4, we show that when dynamic hedging a linear system may be solved to find the hedging weights, if there an finite number of possible jump amplitudes. However, for an infinite number, minimizing the square of the jump risk weighted by the jump amplitude PDF is shown in Appendix A.2.4 to minimize the

variance of the expected jump risk squared. Therefore, dynamic hedging is sometimes referred to as local variance minimization.

We explain how the self-financing condition can be enforced for semi-static hedging in Appendix A.2.5. We formulate the minimization problem and find it to very similar to that of dynamic hedging. Both dynamic and semi-static hedging result in least squares problems, as demonstrated in Appendices A.2.4 and A.2.5. In Appendix A.2.6, we show that the optimal weights $\boldsymbol{\phi}$ satisfy $A\boldsymbol{\phi} = \mathbf{b}$ where A is a ζ by ζ matrix. The solution to this system is our least squares hedging strategy.

3.3 Numerical Experiments for Hedging

3.3.1 Change in Hedged Portfolio Value over 1 Time Horizon

Here we present some numerical illustrations of hedging. In section 3.3.2 we simulate jump-diffusion paths over the time period $[0, u]$ and compare the performance of hedging strategies through the distribution of relative P&L. In this section, we study the risk inherent in each strategy due to asset price movements over one time horizon.

Figures 3.7, 3.8 and 3.9 show the change in value of the portfolio $\Delta_J\Pi$ over a single time horizon. In each Figure, the three columns of plots correspond to three different hedging strategies, A,B,C, given by

A. Gauss-Hermite quadrature (the gamma method);

B. least squares weighted by the transition PDF;

C. least squares weighted by the jump amplitude PDF.

The change in portfolio value is

$$\Delta_J\Pi = \Delta_J V - \boldsymbol{\phi}.\Delta_J\mathbf{I},$$

where here we have $\Delta_J\psi = \psi(S) - \psi(S_0)$ and $S_0 = 1$. In Figures 3.7 and 3.8, the period of time over which the asset price moves from S_0 to S is $u = 0.25$, the maturity of the hedging options. Therefore, these Figures demonstrate semi-static hedging. Conversely, in Figure 3.9, the period of time is an instant. Therefore, this Figure illustrates dynamic hedging where the rebalance interval is very small.

In the top plots of each Figure, 0 hedging instruments is delta hedging rather than using A, B or C. If the asset price remains in a small region near $S = 1$, delta hedging is profitable. In agreement with the definition of delta hedging $\frac{\Delta_J\Pi}{\partial S} = 0$ at $S = 1$.

The ideal strategy has $\Delta_J \Pi = 0 \; \forall \; S$, where the hedging options perfectly replicate the target option. Since we are using a finite number of hedging options with maturity $u < T$, this is impossible. Under the strategies A, B, C, as the number of options increases, the curves tend to the horizontal line $\Delta_J \Pi = 0$. The non-smooth points on the curves correspond to the strikes of the hedging options.

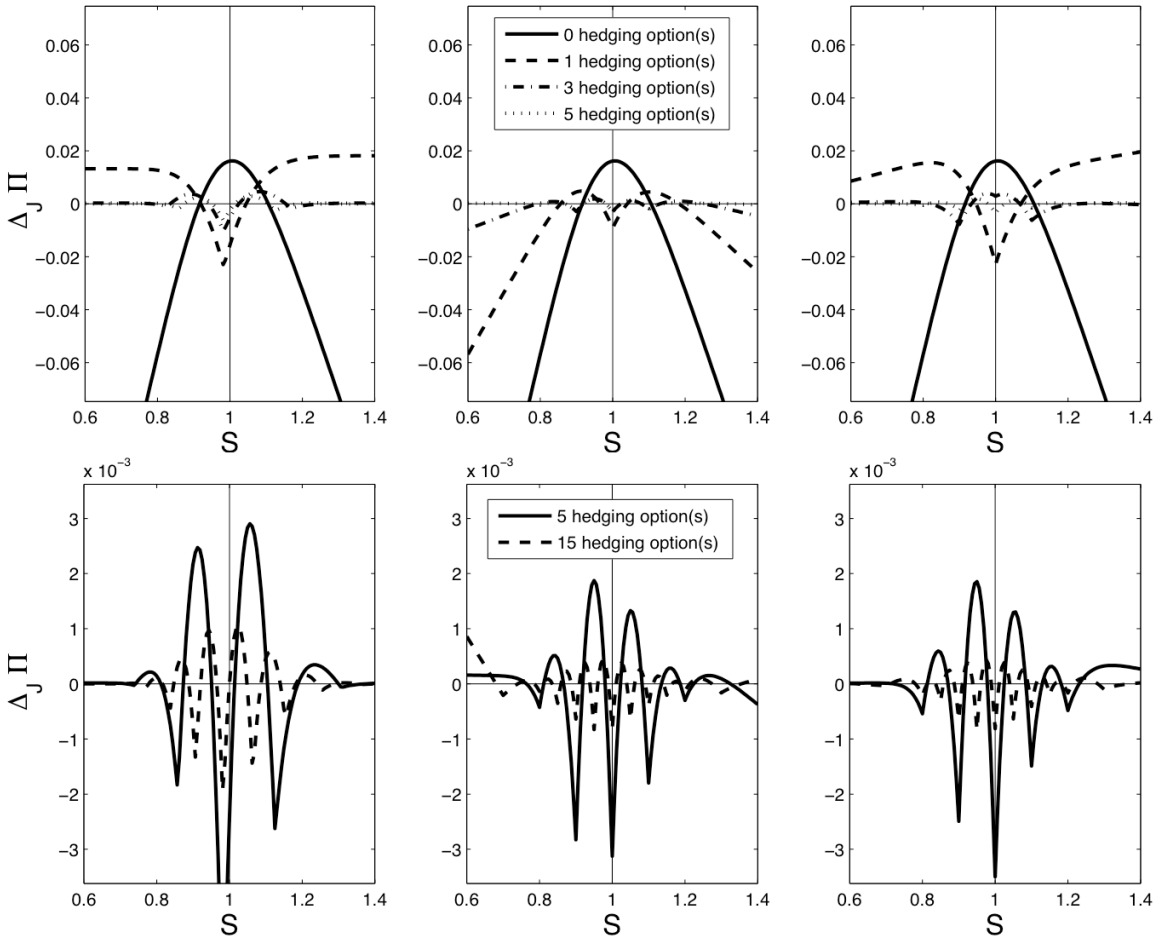


Figure 3.7: The change in portfolio value $\Delta_J \Pi$ under Black-Scholes when the asset price moves from $S_0 = 1$ to S over the time period $u = 0.25$. The target option matures at $T = 0.5$ and the hedging options mature at u. All options are puts. The leftmost plots use Gauss-Hermite quadrature, the central plots use least squares with the transition PDF and the rightmost plots use least squares with the jump amplitude PDF. The plots on the top have 0, 1, 3 and 5 hedging options and those on the bottom have 5 and 15. The bottom plots show a much smaller range of $\Delta_J \Pi$.

Delta hedging is the worst performing strategy of those we study for large moves in asset price. Comparing the strategies under Black-Scholes in Figure 3.7, A results in curves fairly close to $\Delta_J \Pi = 0$. For few hedging options it performs better than B and C, but for a large number of hedging options the opposite is true. This is

because the strikes of the hedging options are chosen by A, whilst the strikes are pre-prescribed for B and C. Indeed we have allowed A to use any strikes, but in reality only a limited set is available, which would weaken the success of the strategy. [6] proposes computing the optimal strikes and weights and then choosing the nearest available strikes; this would have a detrimental effect on strategy A's performance.

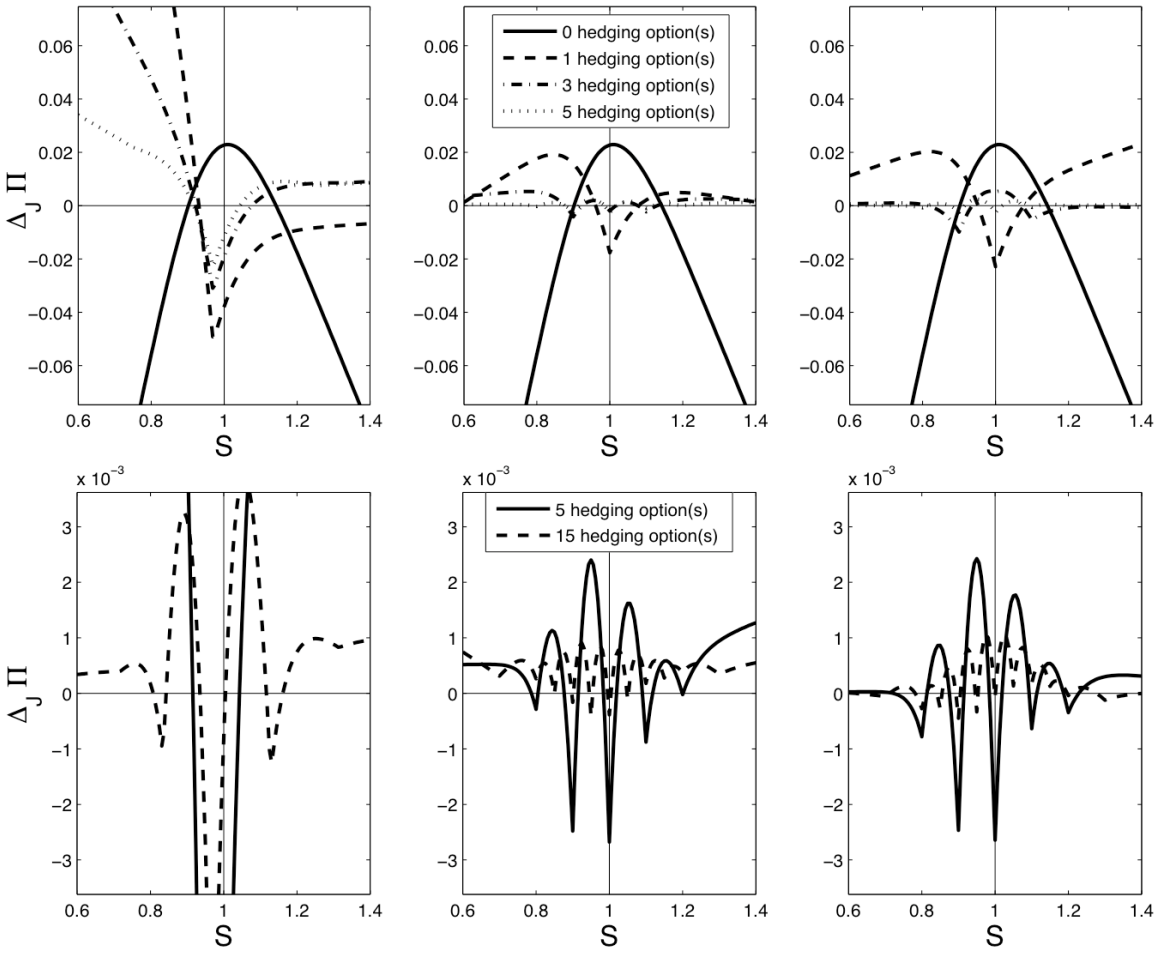


Figure 3.8: This is identical to Figure 3.7, except the asset price evolution is Merton's jump-diffusion. The value of the target at time 0 is 0.0584 (to 3 significant figures).

Moving into Merton's jump-diffusion world, Figure 3.8 demonstrates that A is far less successful than B and C, since $\Delta_J \Pi$ visibly fluctuates in S with much greater amplitude for A. This is because strategy A approximates Merton's weighted sum of Gaussian distributions with one Gaussian distribution and therefore does not capture the full complexity of the transition PDF. There is very little difference in the performance of B and C in Figure 3.8. If anything, strategy B is better for fewer options, because the weighting PDF puts greater emphasis on the portion of the domain $S > 1$ than the weighting PDF of C. As the number of hedging options ζ tends

to ∞, $\Delta_J\Pi$ for strategies B and C converges, at least for S near 1. This is because there is a unique linear combination of hedging options that perfectly replicates the target option and as $\zeta \to \infty$ all "consistent" strategies should converge to this limit.

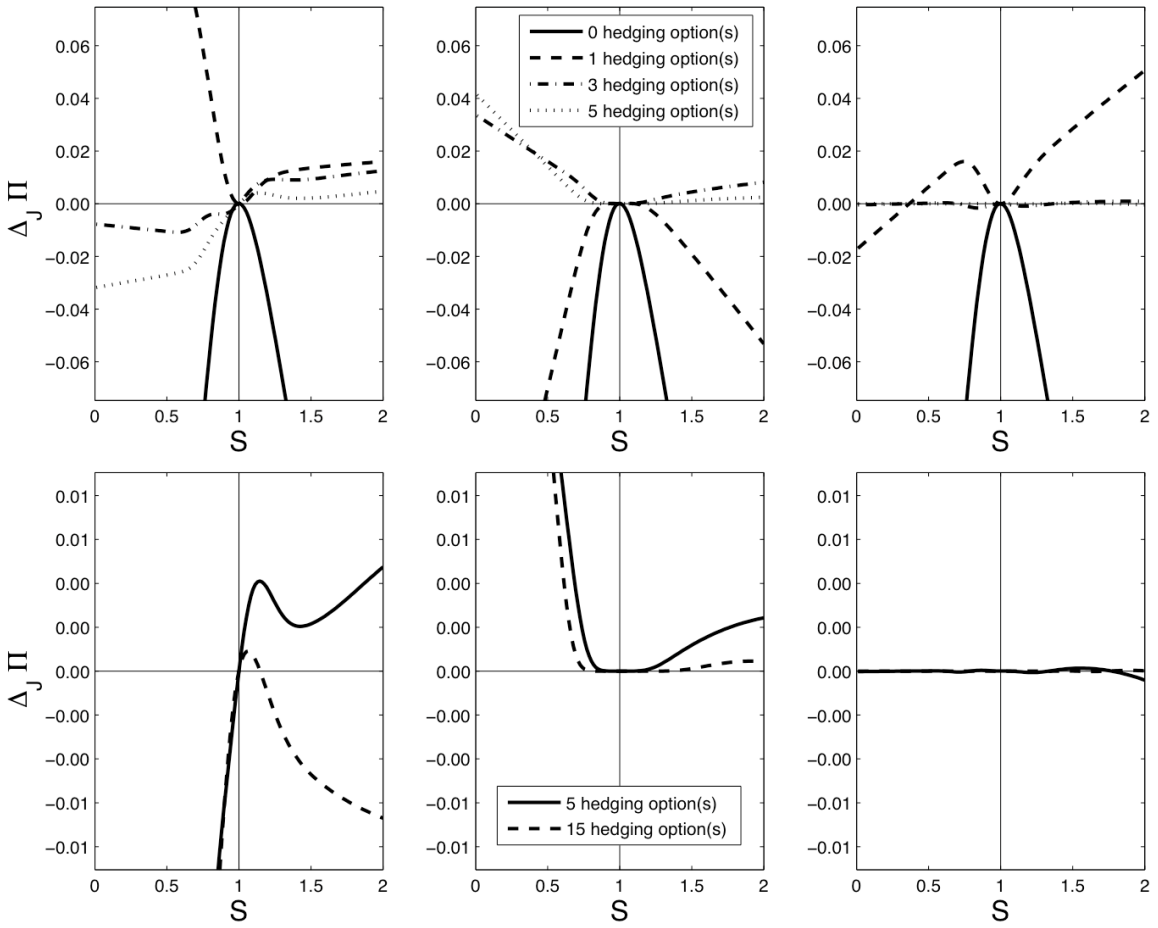


Figure 3.9: This is identical to Figure 3.8, except the period of time over which S moves from $S_0 = 1$ to S is almost instantaneous (exactly $u/2^8 \approx 1/1000$ of a year). This represents the jump risk. The S domain is larger than in Figures 3.7 and 3.8.

When hedging instantaneously, Figure 3.9 illustrates that C is better than both A and B. The S domain is wider in Figure 3.9, because a jump in asset price could be large in magnitude. Strategy C is far less risky if a jump occurs, especially if it has an extreme amplitude. Indeed, for 15 hedging options, C seems to almost perfectly replicate the target option in the region shown, corresponding to the curve with $\Delta_J\Pi \approx 0$. Notice that $\partial\Delta_J\Pi/\partial S = 0$ at $S = 1$ for C in Figure 3.9, because delta neutrality is imposed. Interestingly, without imposing delta neutrality explicitly, B also appears delta neutral. Every curve intersects the point $(0, 0)$, because the hedge cost is equal to the value of the target option at $t = 0$.

3.3.2 Relative Profit and Loss Distributions

Here we present results for the distribution of relative P&L, defined by (3.3). In the previous section, it is clear that the Gauss-Hermite quadrature strategy is worse performing than least squares. Therefore, we conduct experiments for semi-static and dynamic hedging using least squares strategies. The minimization problem is weighted using either the transition PDF or the jump-amplitude PDF. We vary both the number of rebalance times N and the number of hedging options ζ. The target option expires at $T = 0.5$ and hedging options expire at $u = 0.25$. Both are European puts. In all cases, the ideal P&L distribution is a Dirac delta function at the origin.

Figure 3.10 compares semi-static least squares hedging strategies weighted using the transition PDF. All the distributions have PDF ≈ 0 for $S \notin [-0.32, 0.34]$. By no arbitrage, any self-financing hedging strategy cannot be consistently profitable; if it were true, the target option would be too highly priced. Therefore, the mean of each distribution is very close to zero, implying $\int_0^\infty \text{PDF}(S) \cdot S\text{d}S \approx \int_{-\infty}^0 \text{PDF}(S) \cdot S\text{d}S$. For $\zeta = 1$, two peaks for positive relative P&L are balanced by weight towards the left hand (negative) tail of the distribution. $\zeta = 1$ is most dislike the Dirac delta function and therefore appears to be the worst performing strategy of those in Figure 3.10. The best performing seem to be $\zeta = 3$ and $\zeta = 5$. However, a single bin size for the relative P&L is not sufficient to fully interpret the nature of the distribution: the bin size is 4 times smaller on the right providing a finer resolution.

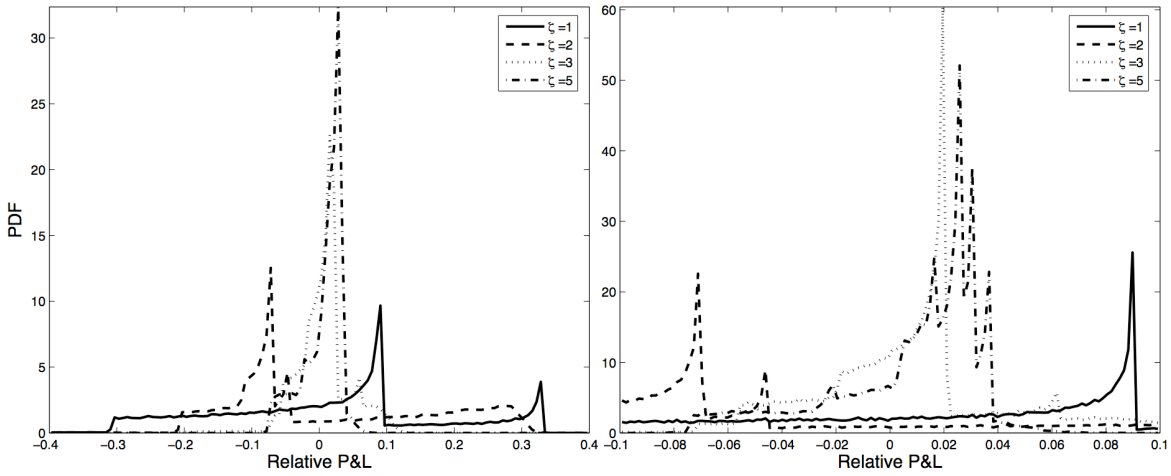


Figure 3.10: The distribution of relative P&L for the least squares strategy weighted with the transition PDF. The number of hedging options is different for each curve: $\zeta = 1, 2, 3, 5$. The plots are of the same distribution, but have different scales in S. Each plot has 2^7 bins over the domain shown. There are $M = 100,000$ simulations, or equivalently, data points.

We may gain valuable insight into the distribution by studying statistics on the underlying data. Table 3.1 provides such summary statistics for the data of Figure 3.10. The smaller in magnitude each statistic, the better the performance of the corresponding hedging strategy. The absolute value of each statistic is almost unanimously decreasing in ζ and therefore, the greater ζ, the better the hedging strategy. The greatest improvement in performance is seen between $\zeta = 2$ and $\zeta = 3$. This is most likely due to the strikes that are used: the first ζ strikes of $\{1, 0.9, 1.1, 0.8, 1.2, 0.7, 1.3, 0.6, 1.4, 0.5\}$. The addition of the $K = 1.1$ hedging option improves the hedge significantly. The lack of improvement from $\zeta = 5$ to 10 is because the target option's gamma[3] (or curvature) at $t = u$ is small for large or small S and therefore the deep in and out of the money hedging options (K large or small) are used sparingly.

Statistical Measure	$\zeta = 1$	$\zeta = 2$	$\zeta = 3$	$\zeta = 5$	$\zeta = 10$
Mean	0.006623	0.007005	0.006929	0.006998	0.006996
Standard Deviation	0.168671	0.175173	0.039895	0.028075	0.026206
Skew	-0.000067	-0.003039	-0.000028	-0.000026	-0.000022
Kurtosis	0.002063	0.004922	0.000016	0.000002	0.000002
0.01%	-0.603985	-0.926533	-0.228105	-0.081117	-0.073473
0.10%	-0.517658	-0.844798	-0.199724	-0.074831	-0.072963
1.00%	-0.306174	-0.614347	-0.119705	-0.070648	-0.068969
5.00%	-0.271240	-0.189065	-0.052725	-0.056462	-0.052188
95.00%	0.312082	0.278211	0.080153	0.036400	0.037135
99.00%	0.328426	0.301659	0.103601	0.047812	0.039005
99.90%	0.329104	0.319212	0.121154	0.065365	0.039099
99.99%	0.329111	0.332545	0.134487	0.078698	0.042452

Table 3.1: Statistics for the distribution of relative P&L for the least squares hedging strategy using the transition PDF. The data is the same as in Figure 3.10. All values are rounded to 6 decimal places. The $x\%$ statistical measure is defined such that $x\%$ of the data is smaller than the statistic given. There are $M = 100,000$ data points.

Hedging options are used in a jump-diffusion world to reduce the impact of jumps in asset price. Jumps of an extreme magnitude may result in very large or very small relative P&L, but occur too infrequently to be visible on plots of the PDF. The percentile and fractional percentile statistics in Figure 3.1 reveal the value at risk with a confidence level. The more options used, the smaller the magnitude of these statistics and therefore, the smaller the risk in the portfolio. The most remarkable statistic is the thousand fold reduction in the kurtosis from $\zeta = 1$ to $\zeta = 10$. This is

[3]Section 3.2.1 details how to determine the hedging option weights using the gamma of the target.

probably due to the outliers and since the experiments make a series of approximations (for speed of execution) and are not exact, the outliers may not be reliable.

Figure 3.11 illustrates the performance of least squares weighted with the transition PDF for different values of N, the number of rebalancing intervals. When $\zeta = 2$, $N = 1$ seems to perform worse than $N = 2$ and $N = 3$, because the distribution of the former has a prominent peak for negative relative P&L. Delta hedging[4] performs best as $N \to \infty$. For a very small number of hedging options, the least squares hedge is a small improvement on delta hedging and therefore a slightly larger value of N than 1 improves performance.

However, for $N = 8$, the distribution is hardly visible and therefore the performance has deteriorated massively. This is probably due to the fact that the transition PDF does not have sufficient weight for small S, to which the asset price is likely to move if there is a jump. Therefore, using the transition PDF for the weighting when dynamically hedging is far from optimal. This is evident in the right hand plot. We note the uncanny similarity between the shape of the distribution for $N = 1$ in the two plots, albeit translated, which we may only speculate is due to $\Delta_J \Pi$ having a similar shape for both hedging strategies where S is near 1, bar a constant difference.

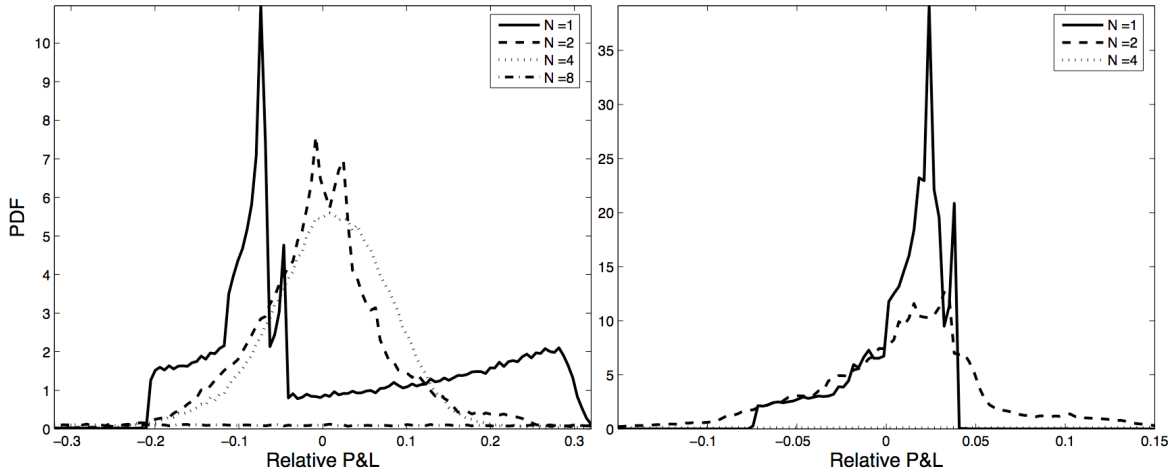


Figure 3.11: The distribution of relative P&L for the least squares strategy weighted with the transition PDF. The number of rebalance intervals is varied, from semi-static to dynamic. In the left plot, $\zeta = 2$ hedging options are used and in the right plot $\zeta = 10$. Each plot has 2^7 bins over each domain shown.

[4]See section 3.1.1 for a full explanation of delta hedging and accompanying experiments.

Finally, we use least squares weighted with the jump-diffusion PDF. Figure 3.12 demonstrates the effect of varying N on the left and varying ζ on the right. In the left plot, it appears that the performance improves with increasing N. However, the several-peaked nature of the distributions complicates this idealized picture. Some might argue that $N = 2$ seems to have the most weight near relative P&L $= 0$. In the right hand plot, it appears that $\zeta = 2$ may be the best performing strategy, but all else being equal, one would expect the performance to improve with increasing ζ. Overall, we cannot make strong statements about dynamic hedging from these plots.

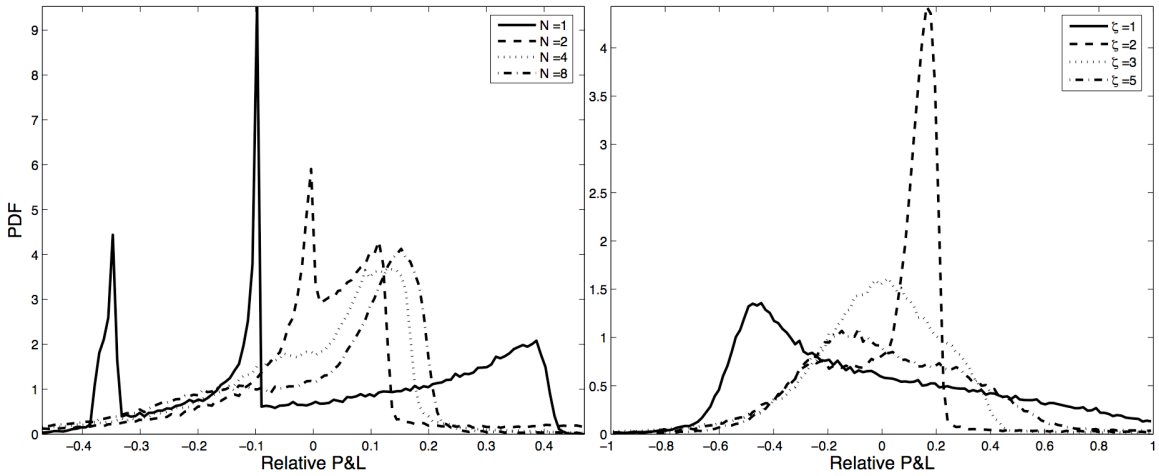


Figure 3.12: The distribution of relative P&L for the least squares strategy weighted with the jump-amplitude PDF. On the left, the number of rebalance times N is varied, with the number of hedging options fixed at $\zeta = 2$. On the right, the number of hedging options is varied with a fixed number $N = 2^5$ of rebalance intervals. Each plot has 2^7 bins over the domain shown.

Unfortunately, producing a table of statistics for dynamic hedging, analogous to Table 3.1 proved troublesome. In particular, a small number of extreme outliers with very large in magnitude relative P&L (of order 10^3) affected the statistics greatly. These outliers probably occurred due to numerical approximations taken during the experiments, which may have resulted in dividing by a near zero number. Using a different weighting for dynamic hedging may produce more pleasing results, because our jump amplitude PDF has very little weight for $S > 1$. For example, the near uniform weighting in the left hand plot of Figure 3.5 may be more appropriate.

Chapter 4

Model Refinement

4.1 A Hawkes Process for Jump Arrivals

4.1.1 The Fundamentals of Hawkes Processes

Merton's jump-diffusion uses a Poisson process to model the occurrence of jumps. This process is independent of the jump history, and is one particular type of point process, of which there are many. Self-exciting point processes are relevant to the arrival of jumps, because market intuition states that past jumps make future jumps more likely. Point processes are sufficiently general that some imply the opposite: past jumps make future jumps *less* likely. In either case, point processes allow the memory of jumps to decay in time, which is compatible with market behaviour.

As mentioned in [9] (p.38), a point process has four equivalent descriptions:

- a counting measure;
- a nondecreasing integer-valued step function;
- a sequence of points;
- a sequence of intervals.

We choose the first: given a set A, our measure is $N(A) = \#\{i|t_i \in A\}$, where t_i is an event and $\#$ means cardinality, so $N(A)$ takes integer values. For ease, we use the notation $N_t = N\big((0,t]\big)$ for $t > 0$ and $N_t = -N\big((t,0]\big)$ for $t < 0$, so that the counting measure is nondecreasing. We study a self-exciting Hawkes process, first proposed in [15], which enables the history to affect the future. A Poisson process is the limit of a Hawkes process as memory of the history tends to zero.

Given a set $\mathbb{H}$ containing all event times, the history of the process is (as in [16])

$$\mathcal{H}_t = \{t_j \in \mathbb{H} : t_j < t\}.$$

N_t is non-decreasing and therefore we require that $\lambda > 0 \Rightarrow \nu > 0$ and $\int_0^\infty \gamma(u)\mathrm{d}u < 1$. The intensity function $\Lambda(t, \mathcal{H}_t)$ expresses the instantaneous likelihood of an event occurring ([15]),

$$\Lambda(t, \mathcal{H}_t) = \lim_{\mathrm{d}t \to 0} \frac{1}{\mathrm{d}t} \mathbb{P}\big[t_{N_t+1} \in [t, t + \mathrm{d}t) | \mathcal{H}_t\big]$$

We define some important concepts for Hawkes processes in Appendix A.3.1. The resulting idea is that the intensity for $t > 0$ can be written as [9] (p.491)

$$\Lambda(t, \mathcal{H}_0) = Z_0(t) + \nu + \int_0^t \gamma(t-u)\mathrm{d}N(u), \tag{4.1}$$

$$\text{where} \qquad Z_0(t) = \int_{-\infty}^0 \gamma(t-u)\mathrm{d}N(u)$$

and as $t \to \infty$, $Z_0(t) \to 0$. Furthermore, given two Hawkes processes with the same parameter set, but different initial condition Z_0, the variation distance between the two processes converges to 0 as $t \to \infty$, as proven in [9] (p.491).

4.1.2 An Alternative Description: Cluster Poisson Processes

As explained in [16], any stationary self-exciting process, such as Hawkes, can be equivalently formulated as a Poisson cluster process. An event is a birth, which initiates a Poisson process for further births. Each immigrant from outside the system is a cluster centre and the clusters are mutually independent by construction. Figure 4.1 depicts this graphically. The resulting process is one dimensional, but we may visualize it as the superposition of N_t Poisson processes at time t. For each cluster, the spawning of children is a Poisson process and similarly for the child cluster spawning grandchildren, ad infinitum. Therefore, each birth may also be viewed as a cluster centre.

The first cluster centre represents Z_0, the entire history for $t \in (-\infty, 0]$, and therefore has non-unit weight. As in Figure 4.1, a new unconnected tree takes root when an immigrant from outside the system spontaneously materializes to form a cluster centre. This is a non-self-exciting event due to the underlying base intensity ν, and therefore such first generation events are uniformly distributed in time. A Hawkes process at time t is the combination of all clusters at time t, and the Hawkes intensity at time t is the sum of the intensities of each cluster at time t. The intensity is not an increasing function of $t \ \forall \ t \in (0, \infty)$, because the number of births in a cluster is finite with probability one ([9], p.243).

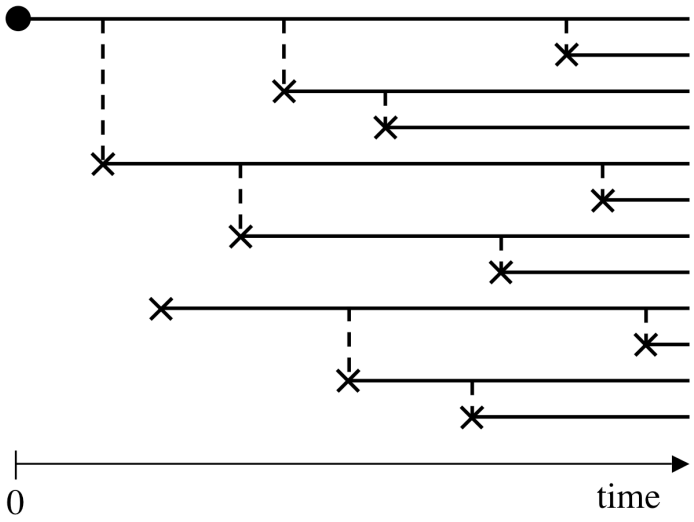


Figure 4.1: A sample family tree for Poisson clusters. The crosses each have unit weight and represent births, or immigrants if at the root of a tree. The dot at $t = 0$ represents the historical input Z_0, which does not in general have unit weight.

Clusters are a more intuitive way to imagine a Hawkes process. For example, [18] uses the Hawkes process to describe the clustering of trades in a market. In a manner analogous to summing the effect of jumps, [18] superposes the impacts of trades on price. However, [18] has two Hawkes processes running simultaneously: one describes the buy-side's executed buy orders and the other their executed sell orders.

Similarly, our model of asset price jumps arriving according to a Hawkes process may be improved by using two mutually exciting Hawkes processes: one describing jumps where $Y > 1$ and one where $Y < 1$, or alternatively one for $Y > \mathbb{E}(Y)$ and another for $Y < \mathbb{E}(Y)$. A single Hawkes process would be difficult to calibrate to the market and two would definitely be unfeasible. However, hedging under model mis-specification can be simulated using a variety of experimental parameter sets.

4.1.3 The Distribution of N_t

We derive some fundamental properties of Hawkes processes in Appendix A.3.2. Ideally, we would like to derive the distribution for the number of jumps in $(0, t]$, or equivalently, find the probabilities $P(N_t = k)\ \forall\ k \in \{0\} \bigcup \mathbb{N}$.

The asymptotic process yields an analytical distribution for N_t [16] (p.501):

$$\mathbb{P}\left(\frac{N_t - \frac{\nu t}{1-m}}{\frac{\sqrt{\nu t}}{(1-m)^{3/2}}} \right) \leq y \to \phi(y) \qquad \text{as } t \to \infty. \tag{4.2}$$

Here $\phi(y)$ is the standard normal distribution and $m = \int_0^\infty \gamma(u)\mathrm{d}u$. The result (4.2) is obtained via the Central Limit Theorem; we explain heuristically how this comes about.

Time $[0, t]$ may be divided into L intervals, and as $t \to \infty$ with L constant, the dependency between the intervals becomes negligible in comparison to the change in N_t over each interval. Therefore, this change can be treated as an i.i.d random variable. The distribution approaches normal as the number of variables tends to ∞. In (4.2), $\frac{\nu t}{1-m}$ is the expected value of N_t and $\frac{\sqrt{\nu t}}{(1-m)^{3/2}}$ is the standard deviation. For example, if $\gamma = 0$ and there is no memory, it is a Poisson process. As the mean $\to \infty$, the Poisson distribution tends to a normal distribution.

We are really interested in finding $P(N_t = n)\ \forall\ n$ in the non-asymptotic case, where t may be positive and small. Detailed mathematical working is given in Appendix A.3.3. We find that the probability that k events occur in [0,t] can be decomposed as follows:

$$\mathbb{P}(N_t = k) = \mathbb{P}(N_t = k | N_t \geq k) \left[1 - \sum_{n=0}^{k-1} \mathbb{P}(N_t = n) \right].$$

where we can theoretically find $\mathbb{P}(N_t = k | N_t \geq k)$ through repeated integration, although a succinct analytical expression is not forthcoming. However, as given in Appendix A.3.3, the probability of no events occurring in $(0, t]$ is

$$\mathbb{P}(N_t = 0) = e^{-(\omega + \nu t)}$$

where $\omega = \int_0^t Z_0(u)\mathrm{d}u$. Note that for the parameter values we choose in section 4.2.2, $\omega << \nu t$. We speculate that in general the form is

$$\mathbb{P}(N_t = k) = e^{-(\omega+\nu t)} \sum_{i=0}^{k} \psi_i t^i$$

where each ψ_i is some function of ν, ω. We further postulate that

$$\mathbb{P}(N_t = k) \to a e^{-bk} \qquad \text{as } \nu \to 0,$$

with a, b constant.

4.2 Hawkes with Exponential Decay

4.2.1 Exponentially Decaying Memory

A typical form for the memory $\gamma(t)$ is exponential decay. It is reasonable to assume the memory decays according to $\mathrm{d}\gamma/\mathrm{d}t = -\beta\gamma$ where $\beta > 0$ is some rate of decay. Furthermore, using exponential decay leads to tractable models. Therefore, we let

$$\gamma(t) = \alpha e^{-\beta t} \tag{4.3}$$

with α, $\beta > 0$ constants and substituting into (A.15) gives

$$\lambda = \frac{\nu}{1 - \frac{\alpha}{\beta}}.$$

We may substitute (4.3) into the intensity function. We do so in Appendix A.3.4 and obtain the the intensity function after n events:

$$\Lambda(t) = \nu + Z_0(t) + \alpha \sum_{j=1}^{n} e^{-\beta(t-\tau_j)} \tag{4.4}$$

where $\alpha = \beta(1 - \nu/\lambda)$ and

$$Z_0(t) = \frac{\lambda\alpha}{\beta} e^{-\beta t}.$$

We derive some fundamental properties for exponential decay in Appendix A.3.5.

4.2.2 Simulating Hawkes Processes

Using (4.4) we may simulate a self-exciting Hawkes process. We cannot compute the event times as easily as we may for a Poisson process, because $\Lambda(t)$ is non-constant. Using a uniform discretization of time $(0, T)$ with interval $\mathrm{d}t$, at each time level, we draw a random uniform $U \in \mathcal{U}[0, 1]$. If $U \leq \mathrm{d}t\Lambda(t)$, an event is said to occur and N_t is incremented. As $\mathrm{d}t \to 0$, the simulation will tend to a Hawkes process.

Figure 4.2 shows M simulations where λ, ν, β and α are constant for each plot. Each vertical line represents a simulation timeline and a dot represents an event. The right hand plot is a Poisson process, where events are uniformly distributed in the (simulation, time) space. There is positive correlation between history and the future when $\nu < 0.1$; in the left plot ($\nu = 0.04$) events seem to occur in clusters, because events heighten the intensity. The average number of events is identical for any ν, $\mathbb{E}[N_T] = \lambda T = 3$, but the variance of N_T is a decreasing function of ν.

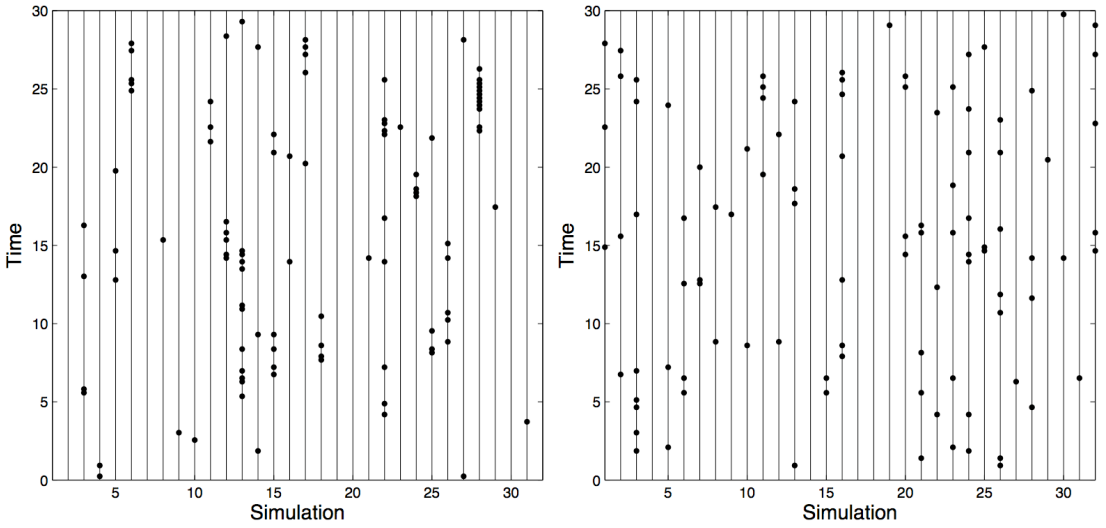


Figure 4.2: Timelines showing simulations for $\nu = 0.04$ on the left and $\nu = 0.1$ on the right. $M = 32$, $T = 30$, $\mathrm{d}t = T/2^7 \approx 0.234$, $\lambda = 0.1$, $\beta = 1$, $\alpha = \beta(1 - \nu/\lambda)$.

The two plots in Figure 4.3 show realizations of intensity over time. The base intensity ν is clearly visible, whilst a 'spike' occurs after an event. To ensure that $\int_0^t \Lambda(u)\mathrm{d}u \to \lambda t$ as $t \to \infty$, the 'spikes' are taller for smaller ν. In addition, events cluster more for smaller ν resulting in 'spikes' combining to momentarily heighten the intensity further. The intensity is almost exactly ν most of the time, but this need not be the case if the decay of the 'spikes' were shallower, due to smaller β.

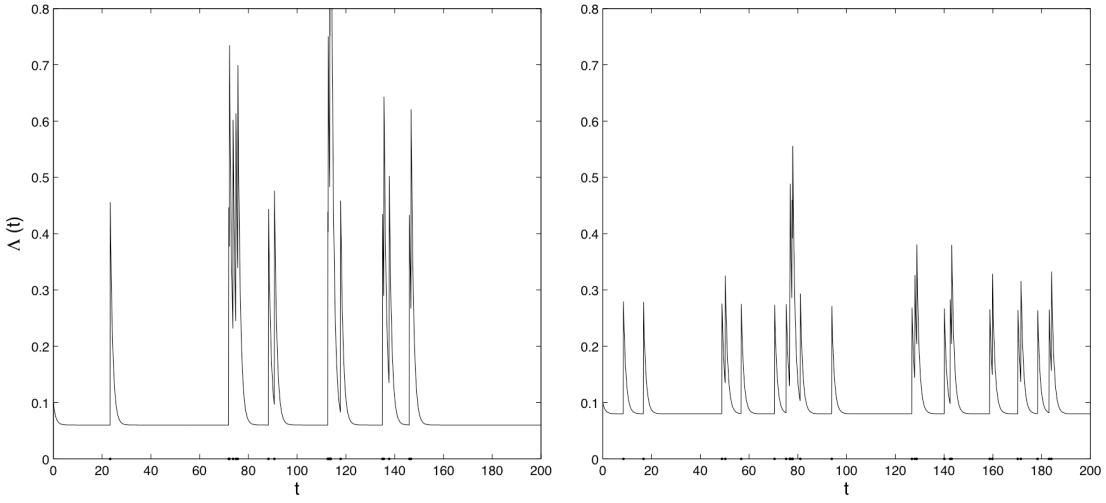


Figure 4.3: The intensity function $\Lambda(t)$ for two simulations: $\nu = 0.06$ on the left and $\nu = 0.08$ on the right. $T = 200$, $\mathrm{d}t = T/2^{11} \approx 0.0977$, $\lambda = 0.1$, $\beta = 1$, $\alpha = \beta(1-\nu/\lambda)$. $\mathbb{E}\big[\int_0^T \Lambda(t)\mathrm{d}t\big] = \lambda T = 20$, whilst in these particular realizations, there happen to be 19 events on the left and 26 on the right. The dots on the t axis represent events.

Figure 4.4 shows the distribution of N_t, the number of events occurring during each simulation. For all values of ν, the mean is $\lambda T = 3$. When $\nu = 0.1$, the point process is Poisson and the simulations match the analytical Poisson distribution fairly well. As ν decreases, the distributions have less weighting at the mean and a greater weighting in the tails. Specifically, the probability of zero events and a large number of events both increase. Therefore, the mean is a poor measure of the distribution.

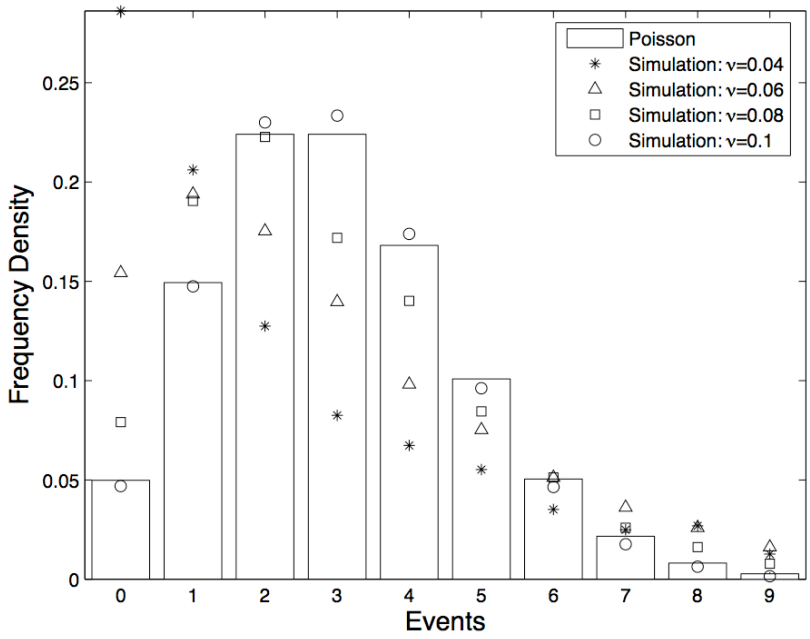


Figure 4.4: The distribution of N_t for each simulation for different values of ν against the Poisson distribution. This is a Monte Carlo estimate for the distribution of N_t. $M = 2^{11}$, $T = 30$, $\mathrm{d}t = T/2^7 \approx 0.234$, $\lambda = 0.1$, $\beta = 1$, $\alpha = \beta(1 - \nu/\lambda)$.

4.2.3 Implications for Pricing and Hedging

If Merton's jump-diffusion is adapted so that a Hawkes process drives the jumps, rather than a Poisson process, the price of options changes. Firstly, the parameters ν and β need to be calibrated with the market. The distribution of events most likely has an analytical formula, analogous to the Poisson probability. These probabilities need to be used instead of the Poisson probability wherever it occurs in series solutions for European option prices and Greeks (sensitivities).

In terms of hedging options, static hedging changes, due to a new transition PDF, where the weights are based on a Hawkes process distribution rather than Poisson. For dynamic hedging, the distribution of jump amplitudes is unchanged, so we may hedge as under a Poisson process. However, when a jump occurs, the likelihood

of jumps in the near future increases and when a jump does not occur, likelihood decreases. We can adapt our hedge to better reflect this changing situation. For example, if $\Lambda(t) > \lambda$, we should place added emphasis on hedging jumps.

Mathematically, the instantaneous change in Λ is given by

$$\mathrm{d}\Lambda_t = \alpha \mathrm{d}N_t - \gamma(\Lambda_t - \nu)\mathrm{d}t.$$

In Appendix A.3.6, we derive the PIDE for Merton's jump-diffusion with Hawkes:

$$\begin{aligned} 0 =& V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + (r - \lambda\kappa)SV_S - rV - \beta(\lambda - \nu)\frac{\partial V}{\partial \Lambda} \\ &+ \lambda\left(\int_0^\infty V(YS, t, \Lambda + \alpha)g(Y)\mathrm{d}Y - V(S, t, \Lambda) \right) \end{aligned}$$

where $g(\cdot)$ is the jump amplitude distribution.

A better model of asset price evolution may claim that the intensity is a function of the historical jump amplitudes: large jumps trigger more jumps than smaller ones. The historical dependence function needs to be altered to reflect this. For example,

$$\Lambda(t, \mathcal{H}_t) = \nu + \int_{-\infty}^{t} |Y(u) - 1|\gamma(t - u)\mathrm{d}N(u) \tag{4.5}$$

where $Y(u) = 0$ if a jump has not occured at $t = u$ and $Y(u)$ is the jump amplitude at $t = u$ if a jumps occurs then $(u = \tau_j)$. Note that analytical formulae for European option prices may not exist and determining how to hedge would be more complicated.

4.2.4 Hedging Experiments for a Hawkes Process

Here we conduct hedging experiments to compare with those in section 3.3.2. Instead of jumps occurring with probabilities of the Poisson distribution, $\mathbb{P}(N_t = n)$ is computed through Monte Carlo simulation of a Hawkes process with case $\nu = 0.01$, $\beta = 10$. See Appendix A.3.7 for an explanation and a table of the probabilities used.

We semi-static hedge (no rebalances) blind to the fact that the jump arrivals are governed by a Hawkes point process. Therefore, hedging is conducted *as if* the asset price evolution has Poisson distributed jumps, but the simulated paths actually have jumps driven by a Hawkes process. Table 4.1 displays the relative change in each statistic from Table 3.1, for which analogous experiments were conducted. The statistics themselves refer to the distribution of relative profit and loss (P&L). Let the statistic under Hawkes arrivals be x_H and the statistic under Poisson arrivals be x_P, then each 6-decimal point number represents

$$\frac{x_H - x_P}{x_P}.$$

This enables us to examine the relative change (percentage change) in each statistic. In Table 4.1, notice that the skew is much larger for a small number of hedging options. This is because there is greater probability that more than one jump will occur with a Hawkes process, and since the expected jump size is large and negative, this usually results in a large downward movement in asset price. If only one at-the-money option is used for hedging, then the hedge is inadequate for small S and large losses occur.

Statistical Measure	$\zeta = 1$	$\zeta = 2$	$\zeta = 3$	$\zeta = 5$	$\zeta = 10$
Mean	0.205345	0.882512	0.135662	0.099743	0.044740
Standard Deviation	0.011727	-0.030284	0.001679	-0.015922	0.006754
Skew	12.865672	0.141823	1.214286	0.000000	0.090909
Kurtosis	0.353854	0.282202	0.750000	0.000000	0.000000
0.01%	0.285285	0.176081	0.248351	0.154665	0.153049
0.10%	0.486149	0.282048	0.414257	0.244872	0.150528
1.00%	-0.006735	-0.663495	-0.413650	0.013065	0.019429
5.00%	-0.011709	-0.041568	-0.044267	-0.029666	0.020541
95.00%	0.001647	-0.000234	-0.000811	0.000302	0.001212
99.00%	-0.000192	-0.001883	-0.005483	-0.011880	0.000359
99.90%	-0.000003	-0.002522	-0.006644	-0.012315	0.000000
99.99%	0.000000	-0.002273	-0.005621	-0.009606	-0.017808

Table 4.1: The relative *change* in the statistics for the distribution of relative P&L for the least squares hedging strategy using the transition PDF. The hedging strategy assumes the counting process is Poisson, but it is actually Hawkes with $\nu = 0.01$ and $\beta = 10$. All values are rounded to 6 decimal places. The percentages mean that $x\%$ of the data is smaller than the value given. There are $M = 100,000$ data points.

Referring to Table 4.1, large positive values imply a worsening of performance, whilst large negative values can be interpreted as an improvement. The mean has increased slightly, which is fairly immaterial. The kurtosis, 0.01% and 0.10% statistics have also deteriorated slightly. However, the changes are not substantial. The table summarizes the effect of one type of model mis-specification, misjudging the exact nature of the counting process for jumps, which is highly likely due to the difficulty in calibration. Since the effect is small, we can conclude that misjudgment of the counting process hardly affects the performance of the hedge. To add, improving hedging performance in increasing ζ remains. Hedging with just a few options is substantially better than delta hedging. However, in order to make further conclusions on mis-specification, it would be beneficial to use a larger sample size than $M = 100,000$ paths and to vary other parameters, such as λ.

Chapter 5

Conclusion

We have studied pricing, hedging and model refinement under Merton's jump-diffusion. In chapter 2 we introduced the model, comparing it with Black-Scholes. We explained in detail how to efficiently simulate asset price paths, making use of the independence of the jumps and Brownian motion. We found Monte Carlo estimation of European vanilla option prices to be consistent with Merton's series solution. We formulated the asset price evolution as a PIDE and solved it numerically. Then we used the penalty method to price American options and obtained quadratic convergence in our solutions. Note that for equivalent accuracy, solving the PIDE is much faster than using Monte Carlo methods and pricing American options is straightforward. However, when extending Monte Carlo methods to high dimensions[1], the computational cost of Monte Carlo becomes smaller than that of PIDE solvers.

In chapter 3 we introduced the concept of hedging and studied the conventional delta hedging strategy. Under Merton's jump-diffusion, this strategy fails to hedge jumps and therefore has a material probability of incurring large losses. Instead, we used options to hedge jumps. If there are a finite number n of possible jump amplitudes, dynamically hedging eliminates all risk if we use at least n options, the underlying and bonds. If there are an infinite number of possible jump amplitudes, then a continuum of strikes (an infinite number) of hedging options perfectly replicates the target. In the market, there are a finite number of strikes and hence hedging errors occur, which we measured the error through the relative profit and loss (3.3).

Our two principal hedging strategies are Gauss-Hermite quadrature and least squares. The latter is formulated as a quadratic minimization problem with a weighting representing our expectation: we used the transition PDF and the jump amplitude PDF. In the Black-Scholes world, Gauss-Hermite quadrature is an adequate

[1] An option whose value is derived from n underlying assets has dimension n.

static hedge. However, under Merton's jump-diffusion, Gauss-Hermite quadrature approximates the price evolution crudely and is therefore not optimal. Furthermore, it is based on the assumption of a continuum of available strikes, which is unrealistic. The Least squares hedging strategy replicates the target better.

Through Monte Carlo simulation of hedging strategies, we find that least squares weighted with the transition PDF is a successful semi-static hedging strategy. As the number of hedging options increases, the strategy performs better on almost every statistical measure we analyze. However, using the strategy for dynamic hedging with frequent rebalancing fails, because the transition PDF does not place enough emphasis on jumps. Least squares weighted with the jump amplitude PDF appears to perform better for dynamic hedging.

In chapter 4 we refined Merton's jump-diffusion model, by replacing the Poisson process for the arrival of jumps by a Hawkes process, a cluster Poisson process. The jump history has an influence on the current intensity. We illustrated some important properties of such processes and initiated thoughts towards uncovering the distribution of jumps. When the memory has the form of exponential decay, a Hawkes process is tractable. Indeed, if the distribution of jumps has an analytical form, then there is a close form price for European options with the Hawkes process, analogous to Merton's series solution. We find that semi-static hedging under a Hawkes process performs well, even though we hedge as if the process is Poisson. Mis-specification of the counting process barely effects the success of the hedge. Indeed, dynamically hedging options under this model may actually perform better than under a Poisson process if we are able calibrate to the Hawkes process, because we have more information about the likelihood of a jump occurring in the next interval.

Some future areas of research are incorporating transaction costs and optimizing over strikes and rebalance times. Leland's transaction costs were first proposed in [25], where a fixed proportion of each transaction is taken as a non-recoverable charge. It is supposed that the percentage transaction cost is the same for all financial instruments. [22] proves that the price of an option tends to a finite limit as the number of rebalances tends to ∞, provided the transaction proportion also tends to zero like $N^{-\alpha}$ with $\alpha \in [0, 1/2]$ and N the number of rebalances. Alternatively, transaction costs may be a fixed absolute amount for each security, based on the bid-ask spread, as detailed in [23]. This may be considerably greater for exotic options than for stocks. Transaction costs justify using a small number of hedging options or a small number of rebalance times or both, since altering the hedge incurs a cost.

To optimize the rebalancing times, there are three principal methods: time interval strategies, where the duration to the next portfolio rebalancing depends upon the current stock price; price change strategies, where rebalancings occur when the stock price changes by pre-described amounts; and renewal strategies, where price-time boundaries are set upon each rebalancing with the subsequent rebalancing occurring when the price trajectory hits the boundary. Under jump-diffusion, price change strategies seem most appealing, because the model is time-homogeneous. Before dynamically hedging, the rebalance rules can be set so that the expected number of rebalance times is N, where N is chosen to balance transaction costs with the hedging error. When hedging jump-diffusion with a Hawkes process, rebalancing could occur when the counting process intensity changes by a pre-prescribed amount. However, for the purposes of numerical simulation, it is faster to have a fixed (and small) number of possible rebalance times, both known prior to runtime, since all price, deltas and least squares solutions can be pre-computed prior to execution.

For least squares, it would be better to minimize over all possible strikes K_i in addition to the hedging weights ϕ_i. Incorporating transaction costs would naturally lead to a finite number of hedging option strikes. Future research could entail optimizing over strikes, weights and rebalance times as well as using different variants on the weighting function. This is because semi-static hedging should be thought of as a subset of dynamic hedging, which needs a particular weighting function chosen from a spectrum of possibilities, suitable to the rebalance time-frame. The resulting strategies should be tested for robustness against model mis-specification. However, the more complicated the hedging, the more computationally demanding the experiments become. We cannot possibly simulate every strategy permutation. It is better to constrain the strategy world and find the optimal solution than to find a non-optimal strategy in an unbounded space.

Appendix A

Appendix

A.1 Supplementary Details for Chapters 1 & 2

A.1.1 The Probability of a Market Crash

This refers to the statement made in section 1.1 concerning the probability of a market crash. Assume there are 2^8 trading days in a year and use a cautious estimate for volatility of $\sigma = 30\%$. The price change is

$$\frac{S_0 + \Delta S}{S_0} = 1 - 0.225 = 0.775$$

and letting $x = \log(S)$ we have $\Delta x = \log(0.775) \approx -0.1106$. The Gaussian probability density function (PDF) is

$$\frac{\exp\left(-\frac{x^2}{2\sigma^2 t}\right)}{\sqrt{2\pi\sigma^2 t}}$$

so we can work out the probability of a fall in price from $x_0 = 0$ to at least $u = \Delta x$. Integrating the PDF, we obtain

$$p(u) = \frac{1}{\sqrt{2\pi\sigma^2 t}} \int_{-\infty}^{u} e^{-x^2/(2\sigma^2 t)} \mathrm{d}x \tag{A.1}$$

The complementary error function is

$$\mathrm{erfc}(x) = \frac{2}{\pi} \int_{-\infty}^{-x} e^{-y^2} \mathrm{d}y \tag{A.2}$$

and through the transformation $y = z/(\sigma\sqrt{2t})$, (A.1) may be related to (A.2):

$$p(u) = \frac{\pi\sigma\sqrt{2t}}{2} \frac{1}{\sqrt{2\pi\sigma^2 t}} \mathrm{erfc}\left(\frac{u}{\sigma\sqrt{2t}}\right) = \frac{\sqrt{\pi}}{2} \mathrm{erfc}\left(\frac{u}{\sigma\sqrt{2t}}\right). \tag{A.3}$$

The asymptotic expansion of (A.2) is

$$\text{erfc}(x) = \frac{1}{\sqrt{\pi}xe^{x^2}}\left(1 + \sum_{n=1}^{\infty}(-1)^n \frac{(2n)!}{n!(2x)^{2n}}\right).$$

For large x, we may truncate the series to give

$$\text{erfc}(x) = \frac{1}{\sqrt{\pi}xe^{x^2}} + \text{O}(e^{-x^2}x^2).$$

Using the approximation for erfc(x) in (A.3) gives

$$p(u) \approx \frac{\exp\left(-\left(\frac{u}{\sigma\sqrt{2t}}\right)^2\right)}{2\left(\frac{|u|}{\sigma\sqrt{2t}}\right)} \tag{A.4}$$

gives $p \approx 3 \times 10^{-9}$ and the probability of such a crash occurring in any given year is approximately $256p \approx 8.3 \times 10^{-7} \approx 10^{-6}$.

Note that some websites claim that the probability of the stock market crash is 10^{-160}. Take the price drop to be $\Delta S/S_0 = 0.29\%$ which was the maximum change during the day, but not the close minus the open. Take the volatility to be 0.2% and approximate the number of trading days by 360, then using $(A.4)$ with the *wrong* value for Δx, gives the *incorrect* probability of approximately 10^{-160}. However, we should actually use $\Delta x = \log(1-0.29) \approx -0.149$, giving a daily probability of 3×10^{-45} under these questionable assumptions.

A.1.2 Dynamic Time-stepping for Solving the PIDE

This refers to the use of a variable time-step for the PIDE solver of section 2.3, as mentioned in [13]. Price as a function of S is almost non-smooth near expiry ($\tau = 0$), because the payoff of a call or put option is not smooth at $S = K$, the strike. Dynamic time-stepping chooses smaller time-steps when the solution is less smooth so that the accuracy may be improved. This is rather like the Milne device [27], a method of varying the time-steps for stiff ordinary differential equations.

[10] suggests the following dynamic time-step selector:

$$\frac{\Delta\tau^{n+2}}{\Delta\tau^{n+1}} = \min_i \left[\frac{\frac{\text{dnorm}}{|V_i^{n+1} - V_i^n|}}{\max(D, |V_i^{n+1}|, |V_i^n|)}\right]. \tag{A.5}$$

The paper uses the constants dnorm $= 0.05$ and $\Delta\tau^0 = 0.05/(2^8 \cdot N_x)$. At each time-step n, we test that $\tau_n - \Delta\tau_n > 0$, else we set $\Delta\tau$ to τ_n, to ensure that the total time $\Delta\tau_0 + \Delta\tau_1 + ... + \Delta\tau_{N-1}$ is equal to T, the maturity.

A.2 Mathematical Working for Chapter 3

A.2.1 Delta Hedging under Black-Scholes is Perfect

Delta hedging in the Black-Scholes world is summarized in section 3.1.1. Here we show that delta hedging does indeed perfectly replicate a target option.

Under the Black-Scholes model the instantaneous change in the asset price is

$$\mathrm{d}S = \upsilon \mathrm{d}t + \sigma \mathrm{d}W$$

and using Itô's lemma (given in [19], p.27) the instantaneous change in the target option's value is

$$\mathrm{d}V = \frac{\partial V}{\partial S} S(\upsilon \mathrm{d}t + \sigma \mathrm{d}W) + \left(\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t}\right)\mathrm{d}t$$

Assuming that the hedging weight in the underlying Δ is constant over an infinitesimally small time period, the change in the portfolio value is given by

$$\begin{aligned} d\Pi &= \Delta \mathrm{d}S + bBr\mathrm{d}t - \mathrm{d}V \\ &= \Delta S\big(\upsilon \mathrm{d}t + \sigma \mathrm{d}W\big) + bBr\mathrm{d}t - \frac{\partial V}{\partial S} S(\upsilon \mathrm{d}t + \sigma \mathrm{d}W) - \left(\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t}\right)\mathrm{d}t. \end{aligned}$$

Because $\Delta = \partial V / \partial S$, cancellations remove the dependence on υ and $\mathrm{d}W$, thus making the stochastic differential equation deterministic. Also, assume that $B = V - aS$, because before the instantaneous change, the portfolio has value zero. Therefore we have

$$\mathrm{d}\Pi = \left(V - \frac{\partial V}{\partial S} S\right) r\mathrm{d}t - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\mathrm{d}t - \frac{\partial V}{\partial t}\mathrm{d}t.$$

If $\mathrm{d}\Pi = 0$, (and dividing by $\mathrm{d}t$) we obtain the Black-Scholes PDE [4]:

$$-rV + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} = 0.$$

Since this PDE is indeed satisfied by V under the Black-Scholes model, $\mathrm{d}\Pi$ does indeed equal zero and given that $\Pi|_{t=0} = 0$, we therefore have $\Pi = 0 \ \forall \ t \in [0, T]$ as required.

A.2.2 Mathematical Working for the Gamma Method

Here we derive a useful expression for the price of the target option in terms of a weighted sum of hedging options. This result is used in section 3.2.1.

In 1978, Breeden and Litzenberger showed that the probability density function q under the risk-neutral measure can be expressed as

$$q(S,t;K,T) = e^{r(T-t)}\frac{\partial^2 V}{\partial K^2}(S,t;K,T). \tag{A.6}$$

where V is the target option value, with $(S,t;K,T)$ corresponding to (asset price at initial time, initial time; asset price at future time, future time). We write the value of an option at time t $(t<u)$ as the discounted expected value it will have at time u,

$$V(S,t;K,T) = e^{-r(u-t)}\int_0^\infty q(S,t;\mathcal{K},u)V(\mathcal{K},u;K,T)\mathrm{d}\mathcal{K}. \tag{A.7}$$

The integrand is the value given that $S(u)=\mathcal{K}$, multiplied by $\mathbb{P}[S(u)=\mathcal{K}|S(t)=S]$. As in [6] (p.8), we substitute (A.6) into (A.7), integrate by parts twice and either assume the intuitive European call (C) or European put (P) boundary conditions:

$$\lim_{K\to\infty} C = 0, \quad \lim_{K\to\infty}\frac{\partial C}{\partial K} = 0, \qquad \lim_{K\to 0} P = 0, \quad \lim_{K\to 0}\frac{\partial P}{\partial K} = 0,$$
$$\lim_{S\to 0} C = 0, \quad \lim_{S\to 0}\frac{\partial C}{\partial S} = 0, \qquad \lim_{S\to\infty} P = 0, \quad \lim_{S\to\infty}\frac{\partial P}{\partial S} = 0.$$

Note that any payoff may be constructed from a linear combination of calls and puts with various strikes. Therefore, provided the option is European, all boundary terms disappear and we have the integral

$$V(S,t;K,T) = \int_0^\infty w(\mathcal{K})V(S,t;K,u)\mathrm{d}\mathcal{K}$$

where the weighting function is given by

$$w(\mathcal{K}) = \frac{\partial^2 V}{\partial \mathcal{K}^2}(\mathcal{K},u;K,T). \tag{A.8}$$

A.2.3 Gauss-Hermite Quadrature Nodes and Weights

We obtain the strikes and weights of Gauss-Hermite quadrature, which are used in section 3.2.1.

The Gauss-Hermite quadrature rule has a small error in approximating integrals with respect to a weighting e^{-x^2} on an infinite interval, such as the Gaussian function. Using only ζ nodes, the error is zero for a polynomial integrand $f(x)$ of degree $2\zeta - 1$. Concretely, the rule satisfies

$$\int_{-\infty}^{\infty} f(x)e^{-x^2}\mathrm{d}x = \sum_{j=1}^{\zeta} w_j f(x_j) + \frac{\zeta!\sqrt{\pi}}{2^\zeta}\frac{f^{(2\zeta)}(\xi)}{(2\zeta)!} \qquad \text{for some } \xi \in (-\infty, \infty).$$

The nodes x_j are the roots of the ζ^{th} Hermite polynomial $H_\zeta(x)$, where the polynomials satisfy the recursive relation

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

where $H_0(x) = 1$ and $H_1(x) = 2x$. Given ζ nodes x_j, the weight associated with each is given by

$$w_j = \frac{2^{\zeta-1}\zeta!\sqrt{\pi}}{[\zeta H_{\zeta-1}(x_j)]^2}$$

A.2.4 Minimizing Jump Risk

Here we try to minimize the portfolio's exposure to a jump in the asset price, following the argument in [17]. The results are relevant to section 3.2.2. Minimizing jump risk is most often applied to dynamic hedging. For ease of notation we neglect to explicitly write dependence on t and S. The instantaneous change in each component of the portfolio in terms of S is given by

$$\begin{aligned}
\mathrm{d}V &= \left[V_t + \frac{\sigma^2 S^2}{2}V_{SS} + \xi S V_S\right]\mathrm{d}t + \sigma S V_S \mathrm{d}Z + \Delta_J V \mathrm{d}J \\
\mathrm{d}\mathbf{I} &= \left[\mathbf{I}_t + \frac{\sigma^2 S^2}{2}\mathbf{I}_{SS} + \xi S \mathbf{I}_S\right]\mathrm{d}t + \sigma S \mathbf{I}_S \mathrm{d}Z + \Delta_J \mathbf{I} \mathrm{d}J \\
\mathrm{d}B &= Br\mathrm{d}t
\end{aligned}$$

where $\xi = r - \lambda\kappa$ and $\Delta_J\psi = \psi(YS) - \psi(S)$ is the instantaneous change due to a jump of amplitude Y. As previously, $\mathrm{d}J$ is the jump process and $\mathrm{d}Z$ is the stochastic

term, representing Brownian motion. Given that our portfolio is $\Pi = -V + \boldsymbol{\phi}.\mathbf{I} + bB$, the instantaneous change in its value is

$$\begin{aligned}
\mathrm{d}\Pi =& -\mathrm{d}V + \boldsymbol{\phi}.\mathrm{d}\mathbf{I} + b\mathrm{d}B \\
=& \quad -\left[V_t + \frac{\sigma^2 S^2}{2} V_{SS}\right]\mathrm{d}t \\
&+ \boldsymbol{\phi}.\left[\mathbf{I}_t + \frac{\sigma^2 S^2}{2}\mathbf{I}_{SS}\right]\mathrm{d}t \\
&+ \big[-\Delta_J V + \boldsymbol{\phi}.\Delta_J \mathbf{I}\big]\mathrm{d}J + bBr\mathrm{d}t \\
&+ \xi S\big[\cancelto{0}{-V_S + \boldsymbol{\phi}.\mathbf{I}_S}\big]\mathrm{d}t + \sigma S\big[\cancelto{0}{-V_S + \boldsymbol{\phi}.\mathbf{I}_S}\big]\mathrm{d}Z
\end{aligned} \tag{A.9}$$

We assume that $\boldsymbol{\phi}$ is constant over $\mathrm{d}t$, because the hedge is not changed until the next rebalancing. If the portfolio is delta neutral then $\partial\Pi/\partial S = 0$. This means that

$$-V_S + \boldsymbol{\phi}.\mathbf{I}_S = 0$$

and therefore the last two terms of (A.9) are zero, as shown by the strike-throughs. In terms of the pricing PIDE for any financial instrument ψ, we may write

$$\psi_t + \frac{\sigma^2 S^2}{2}\psi_{SS} = r\psi + \big[\lambda\mathbb{E}(\Delta_J S) - rS\big]\psi_S - \lambda\mathbb{E}(\Delta_J \psi)$$

where the expectation $\mathbb{E}$ is taken with respect to the risk neutral measure. Therefore, the instantaneous change in the portfolio value may be written as

$$\begin{aligned}
\mathrm{d}\Pi &= r\Pi\mathrm{d}t + \lambda\mathrm{d}t\mathbb{E}[\Delta_J V - \boldsymbol{\phi}.\Delta_J \mathbf{I}] + \mathrm{d}J[-\Delta_J V + \boldsymbol{\phi}.\Delta_J \mathbf{I}]. \\
&= r\Pi\mathrm{d}t + \underbrace{\lambda\mathrm{d}t\mathbb{E}[\Delta_J \Pi]}_{\text{deterministic risk}} - \underbrace{\mathrm{d}J[\Delta_J \Pi]}_{\text{stochastic risk}}.
\end{aligned}$$

The latter two terms represent the instantaneous jump risk. We may force both to zero only by setting $\Delta_J \Pi = 0\ \forall$ jump amplitudes. If there are a finite number $L \in \mathbb{N}$ of possible jump amplitudes, we have a linear system to solve to perfectly hedge the jumps,

$$\begin{aligned}
-\big[V(Y_i S_t) - V(S_t)\big] + \boldsymbol{\phi}.\big[\mathbf{I}(Y_i S_t) - \mathbf{I}(S_t)\big] + S_t\big[Y_i - 1\big] = 0\ \forall\ i \in \{1, ..., L\} \\
-\frac{\partial V}{\partial S}\bigg|_{S_t} + \boldsymbol{\phi}.\frac{\partial \mathbf{I}}{\partial S}\bigg|_{S_t} = 0
\end{aligned}$$

where the second line is for delta neutrality.

However, if there are an infinite number of possible jumps, the continuous version of the above linear system is to solve the minimization problem (3.5). The jump

amplitude PDF $g(\cdot)$ is a candidate for $W(\cdot)$, the weighting function. In this case, the optimization problem is similar to local variance minimization. For a delta neutral portfolio, its instantaneous change may be expressed as

$$\mathrm{d}\Pi = \Delta \mathrm{d}t + \Delta_J \Pi \mathrm{d}J$$

and using $\mathbb{E}[(\mathrm{d}J)^n] = \lambda \mathrm{d}t$,

$$\text{variance}[\mathrm{d}\Pi] \approx \lambda \mathrm{d}t \mathbb{E}\left[(\Delta_J \Pi)^2\right] = \lambda \mathrm{d}t \int_0^\infty [\Delta_J \Pi]^2 g(J) \mathrm{d}J.$$

However, if are unsure of the jump amplitude PDF's real world form, then we could use a different weighting function to reflect this uncertainty, such as the uniform-like weighting function in the left of Figure 3.5.

A.2.5 Minimizing Risk over the Lifetime of Hedging Options

We describe the objective function that we minimize when semi-static hedging. The results are relevant to section 3.2.2. We follow an argument similar to that in [17].

Firstly, assume that our hedge must be self-financing, so initially $\Pi|_{t=0} = 0$. For ease of notation we neglect to explicitly write dependence on t and S. At a rebalance time τ_k, the self-financing equation is

$$\boldsymbol{\phi}_k.\mathbf{I}_k + b_k B_k = \boldsymbol{\phi}_{k-1}.\mathbf{I}_k + b_{k-1} B_k \tag{A.10}$$

where the left hand side represents an instant after rebalancing and the right hand side an instant before. We use the abbreviated notation ψ_k for $\psi(t_k, S_{t_k})$. The value of bonds is determined by $B_k = B_{k-1} e^{r \mathrm{d}\tau}$, where $\mathrm{d}\tau$ is the length of time between rebalancing. The value of the portfolio at time τ_{k+1} before rebalancing is given by

$$\Pi_{k+1} = -V_{k+1} + \boldsymbol{\phi}_k.\mathbf{I}_{k+1} + b_k B_{k+1}.$$

Using a similar expression for Π_k and the self-financing condition (A.10), we have

$$\begin{aligned}\Pi_{k+1} &= -V_{k+1} + \boldsymbol{\phi}_k.\mathbf{I}_{k+1} + e^{r\mathrm{d}\tau}(\boldsymbol{\phi}_{k-1}.\mathbf{I}_k + B_{k-1} e^{r\mathrm{d}\tau} - \boldsymbol{\phi}_k.\mathbf{I}_k) \\ &= -(V_{k+1} - V_k) + \boldsymbol{\phi}_k.(\mathbf{I}_{k+1} - \mathbf{I}_k) + (V_k - \boldsymbol{\phi}_k.\mathbf{I}_k))(e^{r\mathrm{d}\tau} - 1) + e^{r\mathrm{d}\tau}\Pi_k.\end{aligned}$$

We want to minimize the difference between the k^{th} and $(k+1)^{th}$ portfolios, say F,

$$F = \Pi_{k+1} - e^{r\mathrm{d}\tau}\Pi_k = V_{k+1} - e^{r\mathrm{d}\tau} V_k - \boldsymbol{\phi}_k.[\mathbf{I}_{k+1} - e^{r\mathrm{d}\tau}\mathbf{I}_k]$$

and therefore solve the problem

$$\min_{\boldsymbol{\phi}} \mathbb{E}_k[F(\boldsymbol{\phi})^2] \tag{A.11}$$

where $\mathbb{E}_k$ means the expectation taken at rebalance time τ_k. This problem is very similar to (3.5), the problem encountered in dynamic hedging, although the expectation operator is most likely different. If we are certain of the real world behaviour, then we may use the transition PDF for jump diffusion as the weighting function.

When we are hedging a European option with the underlying asset only, or where there is a single jump and we are hedging with the underlying and one option, analytical formulae for the weights that solve (A.11) are given in [8]. These formulae may be *approximated* by delta and gamma hedging. However, in the case of a continuum of strikes, no such analytical formulae are available, although it is likely that a series solution analogous to Merton's option pricing formulae may be found in the future.

A.2.6 Weighted Linear Least Squares

We formulate the weighted least squares problem, which is relevant to section 3.2.2. The solution is used for semi-static and dynamic hedging.

Problem (3.5) may be written as an expectation:

$$\min_{\boldsymbol{\phi}} \mathbb{E}\left[\frac{1}{2}(\Delta_J \Pi)^2\right].$$

This is quadratic in each instrument weight ϕ_k, so we can differentiate the objective function with respect to ϕ_k and set to zero to find the minimum:

$$0 = \frac{\partial \mathbb{E}\left[\frac{1}{2}(\Delta_J \Pi)^2\right]}{\partial \phi_k} = \mathbb{E}\left[\Delta_J \Pi \cdot \frac{\partial(\Delta_J \Pi)}{\partial \phi_k}\right]$$

and recalling that $\Delta_J \Pi = \Delta_J V - \boldsymbol{\phi}.\Delta_J \mathbf{I}$, we have

$$0 = \mathbb{E}\left[\Delta_J \Pi \cdot \Delta_J I_k\right] \quad \forall\ k \in \{1, ..., \zeta\} \tag{A.12}$$

where ζ is the number of hedging options. (A.12) a linear system, to which we may add equality or inequality constraints to the least squares minimization problem. Delta neutrality is one such equality constraint and another is hedging the mean jump size with $E[\Delta_J \Pi] = 0$, as suggested in [1] (p.245). Least squares with equality constraints is solved in [3] and orthogonal regression is analyzed in [28].

Assume that we have chosen a weighting function $W(\cdot)$. To numerically solve the system we must discretize in the jump amplitude direction using nodes and weights

(Y_j,W_j), $j \in \{1,...,N_J\}$, where $W_j \neq W(Y_j)$ in general, because an integration rule is used to obtain the weights, such as quadrature. For the system to be over-determined, we require $N_J > \zeta$. Provided we have the computational budget, we may use a large number of nodes N_J to ensure good accuracy. Discretizing, (A.12) becomes

$$\begin{aligned} 0 &= \int_0^\infty W(Y) \cdot \Delta_J\Pi \cdot \Delta_J I_k \cdot \mathrm{d}Y \\ &\approx \sum_{j=1}^{j=N_J} W_j \cdot \Delta_J\Pi(Y_j) \cdot \Delta_J I_k(Y_j) \quad \forall\ k \in \{1,...,\zeta\} \end{aligned}$$

This can be written in matrix-vector form where each row corresponds to a fixed k:

$$\mathbf{0} = \begin{pmatrix} W_1\Delta_J I_1(Y_1) & \cdots & W_{N_J} I_1(Y_{N_J}) \\ \vdots & \ddots & \vdots \\ W_1 I_\zeta(Y_1) & \cdots & W_{N_J} I_\zeta(Y_{N_J}) \end{pmatrix} \begin{pmatrix} \Delta_J\Pi(Y_1) \\ \vdots \\ \Delta_J\Pi(Y_{N_J}) \end{pmatrix}.$$

Note that Δ_J is an operator and N_J is a constant where the J subscript is *not* an index. Using index notation, the system may be succinctly written as

$$\mathbf{0} = \Delta_J I_{ki} W_{ij} \Delta_J \Pi_j \tag{A.13}$$

where $\Delta_J I_{ki}$ is a ζ by N_J matrix, $W_{ij} = \mathrm{diag}(W_j)$ and $\Delta_J\Pi_j = \Delta_J\Pi(Y_j)$, with implied summation over i,j. Expanding $\Delta_J\Pi_j$, we have

$$\Delta_J\Pi_j = \Delta_J V_j - (\Delta_J I_{lj})^T \phi_l$$

with implied summation over l. Substituting this into (A.13) gives

$$\begin{aligned} & \mathbf{0} = \Delta_J I_{ki} W_{ij} \big(\Delta_J V_j - (\Delta_J I_{lj})^T \phi_l\big) \\ \Rightarrow \quad & \Delta_J I_{ki} W_{ij} (\Delta_J I_{lj})^T \phi_l = \Delta_J I_{ki} W_{ij} \Delta_J V_j \end{aligned}$$

with implied summation over i,j,l. This is of the form $A\boldsymbol{\phi} = \mathbf{b}$ where A is a ζ by ζ matrix. This may be solved via standard numerical linear algebra techniques to obtain $\boldsymbol{\phi}^*$, the optimum instrument weights. Alternatively, [17] suggests that Monte Carlo simulations can be used obtain an estimate for $\boldsymbol{\phi}^*$, although this would be computationally expensive.

A.3 Mathematical Working for Chapter 4

A.3.1 Basic Definitions for Hawkes Processes

Here we give some basic definitions relevant to section 4.1.1.

The intensity function $\Lambda(t, \mathcal{H}_t)$ expresses the instantaneous likelihood of an event occurring ([15]),

$$\Lambda(t, \mathcal{H}_t) = \lim_{\mathrm{d}t \to 0} \frac{1}{\mathrm{d}t} \mathbb{P}\big[t_{N_t+1} \in [t, t+\mathrm{d}t) | \mathcal{H}_t\big]$$

or equivalently

$$\mathbb{P}[\mathrm{d}N_t = 1 | \mathcal{H}_t] = \Lambda(t, \mathcal{H}_t)\mathrm{d}t + O\big((\mathrm{d}t)^2\big).$$

In the case of a self-exciting Hawkes process, the intensity may be written as an integral equation [15],

$$\Lambda(t, \mathcal{H}_t) = \nu + \int_{-\infty}^{t} \gamma(t-u)\mathrm{d}N(u) \tag{A.14}$$

where ν is a constant (base intensity) and $\gamma(t-u)$ defines the dependence on the past, the memory. The average intensity is $\lambda = \mathbb{E}[\Lambda(t)]$ and therefore $\mathbb{E}[\mathrm{d}N(u)] = \lambda \mathrm{d}u$. Substituting this into (A.14) gives

$$\begin{aligned} \lambda &= \nu + \lambda \int_{-\infty}^{t} \gamma(t-u)\mathrm{d}u = \nu + \lambda \int_{0}^{\infty} \gamma(u)\mathrm{d}u, \\ \Rightarrow \lambda &= \frac{\nu}{1 - \int_0^\infty \gamma(u)\mathrm{d}u.} \end{aligned} \tag{A.15}$$

Given a history $\mathcal{H}_0$, the intensity for $t > 0$ can be written as [9] (p.491)

$$\begin{aligned} \Lambda(t, \mathcal{H}_0) &= Z_0(t) + \nu + \int_0^t \gamma(t-u)\mathrm{d}N(u), \\ \text{where} \qquad Z_0(t) &= \int_{-\infty}^{0} \gamma(t-u)\mathrm{d}N(u). \end{aligned}$$

A.3.2 Some Fundamental Properties of Hawkes Processes

Now we look to derive some properties of a Hawkes point process. These results are relevant to section 4.1.3.

Let us refer to the intensity after the i^{th} event since time 0 as the i^{th} intensity, denoted by $\Lambda_i(t)$, where $t > 0$. For example, the zeroth intensity is given by

$$\Lambda_0(t) = \nu + Z_0(t)$$

Now let the mean i^{th} intensity over $[a, b]$ be donated by $\bar{\Lambda}_i[a, b]$ $(t_i < a)$, defined as

$$\bar{\Lambda}_i[a, b] = \frac{1}{b-a} \int_a^b \Lambda_i(t) \mathrm{d}t$$

The probability that there are no jumps in (0,t] is given by

$$\begin{aligned} \mathbb{P}(N_t = 0) &= \exp\big(- t\bar{\Lambda}_0[0, t]\big). \\ &= \exp\left(- t\nu - \int_0^t Z_0(u)\mathrm{d}u \right) \end{aligned} \tag{A.16}$$

This is because each infinitesimally small period of time $[t, t + \mathrm{d}t]$ has Poisson distributed jumps and therefore the probability of no jumps is given by $\exp[-\mathrm{d}t\Lambda(t)]$. If no jumps occur, each period $[t, t + \mathrm{d}t]$ is independent of others and hence we may multiply the probabilities for each period.

Using a similar line of reasoning, the time to the first event after zero has the conditional distribution (as given in [16])

$$\mathbb{P}(Y_0 \leq y_0 | \mathcal{H}_0) = 1 - \exp\left(- \nu y_0 - \int_{-\infty}^0 \big[\Gamma(y_0 - u) - \Gamma(-u)\big] \mathrm{d}N(u) \right)$$

where $\Gamma(\eta) = \int_0^\eta \gamma(u)\mathrm{d}u$. Furthermore [16] claims that the interval distributions can be written down explicitly, which may lead to a method of obtaining the probability distribution for N_t.

As [16] points out, we can find the expected value for N_t,

$$\mathbb{E}[N_t] = \int_0^t f(u)\mathrm{d}u$$

where $f(u) = \mathbb{E}[\Lambda(u)]$.

There are results about the probability generating functional in [16]. These are for the process containing all births up to and including generation n, but this does not help us obtain the distribution for N_t, since grandchildren may be born before children, so the analysis does not respect time.

A.3.3 Mathematical Working for the Distribution of N_t

Here we discuss how one might obtain the distribution of N_t, which is relevant to section 4.1.3.

(A.16) is the probability that there are no events. We may use conditional probability to decompose the next problem,

$$\mathbb{P}(N_t = 1|N_t \geq 1) = \frac{\mathbb{P}(N_t = 1 \bigcap N_t \geq 1)}{P(N_t \geq 1)} = \frac{\mathbb{P}(N_t = 1)}{1 - \mathbb{P}(N_t = 0)}$$

which allows us to find the probability of one event,

$$\mathbb{P}(N_t = 1) = \mathbb{P}(N_t = 1|N_t \geq 1)\big[1 - \mathbb{P}(N_t = 0)\big].$$

We need only compute $\mathbb{P}(N_t = 1|N_t \geq 1)$.

Similarly, for k events,

$$\mathbb{P}(N_t = k|N_t \geq k) = \frac{\mathbb{P}(N_t = k \bigcap N_t \geq k)}{P(N_t \geq k)} = \frac{\mathbb{P}(N_t = k)}{1 - \sum_{n=0}^{k-1} \mathbb{P}(N_t = n)}$$
$$\Rightarrow \mathbb{P}(N_t = k) = \mathbb{P}(N_t = k|N_t \geq k)\left[1 - \sum_{n=0}^{k-1} \mathbb{P}(N_t = n)\right].$$

Therefore, to obtain the full distribution, we need only find the probability that no more events occur after the k^{th} for each $k \in \{0\} \bigcup \mathbb{N}$.

Given that one jump has occurred at $J \in (0, t]$, we may integrate the probability of no jumps after the first over all possible jump times J:

$$\mathbb{P}\big(N_t = 1|N_t \geq 1\big) = \int_0^t \mathbb{P}(N_t = 1|J = u)\mathbb{P}\big(J = u|J \in (0, t]\big)\mathrm{d}u. \tag{A.17}$$

The probability of a jump at a certain time is given by the zeroth intensity function,

$$\mathbb{P}\big(J = u|J \in (0, t]\big) = \frac{\Lambda_0(u)}{t\bar{\Lambda}_0[0, t]} \tag{A.18}$$

and the probability of no more jumps is the Poisson probability,

$$\mathbb{P}(N_t = 1|J = u) = \exp\big((t - u)\bar{\Lambda}_1(u, t]\big). \tag{A.19}$$

We may substitute (A.18) and (A.19) into (A.20) to obtain

$$\mathbb{P}\big(N_t = 1|N_t \geq 1\big) = \int_0^t \exp\big((t - u)\bar{\Lambda}_1(u, t]\big)\frac{\Lambda_0(u)}{t\bar{\Lambda}_0[0, t]}\mathrm{d}u. \tag{A.20}$$

Then we use $\mathbb{P}(N_t = 0)$ given by (A.16) to obtain $\mathbb{P}(N_t = 1)$.

Similarly, for the k^{th} jump, we have

$$\mathbb{P}\big(N_t = k | N_t \geq 1\big) = \int_0^t \mathbb{P}(N_t = k | J_k = u).\mathbb{P}\big(J_k = u | J_k \in (0,t]\big)\mathrm{d}u.$$

where

$$\mathbb{P}(N_t = k | J_k = u) = \exp\big((t-u)\bar{\Lambda}_k(u,t]\big),$$

but the probability distribution for the k^{th} jump time is more complicated. This is a possible area of future research.

An alternative line of thought sees us separate the jump originator into Z_0 and ν, the first being a cluster centre and the latter being the base intensity that may inject immigrants into the system. Then the probability of a single jump is

$$\begin{aligned}\mathbb{P}(N_t = 1) = \quad & \mathbb{P}(1 \text{ jump due to } Z_0)\mathbb{P}(0 \text{ jumps due to } \nu) \\ & + \mathbb{P}(0 \text{ jumps due to } Z_0)\mathbb{P}(1 \text{ jump due to } \nu). \end{aligned} \tag{A.21}$$

However, for 2 jumps or more, this form of decomposition is unwieldy.

A.3.4 Intensity in the Case of Exponential Decay

Here we introduce the intensity function for exponentially decaying memory, which is relevant to section 4.2.1. In terms of the intensity function, substituting exponentially decaying $\gamma(\cdot)$ given by (4.3) into (A.14) gives

$$\Lambda(t) = \nu + \int_{-\infty}^t \alpha e^{-\beta(t-u)}\mathrm{d}N(u).$$

In practice, we have a history of events at certain times in the future. However, for simplicity, assume that $\Lambda(t) = \lambda \ \forall \ t < 0$. This is not possible in actuality, because it would require a uniform distribution of events throughout history, with $N(t+\mathrm{d}t) - N(t) = \lambda \mathrm{d}t \ \forall \ t < 0$. However, for the purposes of simulating the process, it is a convenient form for $\mathcal{H}_0$. This is equivalent to λ/β events at $t = 0$ and no events for $t < 0$. Assuming no events have occurred in $(0,t]$, we have

$$\int_{-\infty}^t \alpha e^{-\beta(t-u)}\mathrm{d}N(u) = \lambda \int_{-\infty}^0 \alpha e^{-\beta(t-u)}\mathrm{d}u = \frac{\lambda\alpha}{\beta}e^{-\beta t}.$$

Now assume there have been n events in $(0,t]$ at times $\tau_1, \tau_2, ..., \tau_n$. The intensity is

$$\Lambda(t) = \nu + Z_0(t) + \alpha \sum_{j=1}^n e^{-\beta(t-\tau_j)}$$

where $\alpha = \beta(1 - \nu/\lambda)$ and

$$Z_0(t) = \frac{\lambda\alpha}{\beta} e^{-\beta t}.$$

A.3.5 Properties of Hawkes in the Case of Exponential Decay

Here we give some properties of Hawkes process when $\gamma(\cdot)$ has the form of exponential decay. These are relevant to section 4.2.1. As detailed in [9] (p.368), the density of the first factorial moment measure for a cluster with centre at the origin is

$$m_1(x) = \delta(x) + \nu\alpha e^{-\alpha(1-\nu)x}$$

If we know the number of clusters, this could lead to an analytical form for N_t.

Given that

$$\int_0^t Z_0(u)\mathrm{d}u = \lambda\alpha\beta^{-2}\big(1 - e^{-\beta t}\big),$$

the mean zeroth intensity over $(0, t]$ is

$$\begin{aligned}\bar{\Lambda}_0(0,t] &= \int_0^t \Lambda_0(u)\mathrm{d}u \\ &= \nu t + \lambda\alpha\beta^{-2}\big(1 - e^{-\beta t}\big).\end{aligned}$$

We may then use (A.16) to obtain the probability of zero events in $(0, t]$. Note that if $\lambda = \nu$ and $\alpha = 0$, we recover the Poisson probability of zero events.

Furthermore, the first intensity is

$$\Lambda_1(t) = \Lambda_0(t) + \alpha e^{-\beta(t-J)}$$

where an event has occurred at $t = J$. Therefore, the mean first intensity over $[J, t]$ is found by

$$\begin{aligned}(t - J)\bar{\Lambda}_1[J,t] &= \int_J^t \Big(\Lambda_0(t) + \alpha e^{-\beta(t-J)}\Big)\mathrm{d}t \\ &= (t - J)\nu + \frac{\lambda\alpha}{\beta^2}\big(e^{-\beta J} - e^{-\beta t}\big) + \frac{\alpha}{\beta}\big(1 - e^{-\beta(t-J)}\big)\end{aligned}$$

where the last term is due to the single jump at J. We may put these results into (A.20) to obtain $\mathbb{P}(N_t = 1)$, but it is easier to use (A.21) to obtain $\mathbb{P}(N_t = 1)$ where

$$\omega = \lambda\alpha\beta^{-2}\big(1 - e^{-\beta t}\big).$$

It is now possible to compare our analytical probabilities $\mathbb{P}(N_t = 0)$ and $\mathbb{P}(N_t = 1)$ with Monte Carlo results of simulations.

A.3.6 Deriving the PIDE for Hawkes Processes

Here we derive a PIDE for Merton's jump-diffusion where the jump arrivals are driven by a Hawkes process. This is relevant to section 4.2.3.

The instantaneous change in Λ is given by

$$\mathrm{d}\Lambda_t = \alpha \mathrm{d}N_t - \gamma(\Lambda_t - \nu)\mathrm{d}t,$$

so when a jump Y occurs we have the following maps: $S \mapsto YS$ and $\Lambda \mapsto \Lambda + \alpha$. The price of the target option is dependent on the intensity, so $V = V(S, t, \Lambda)$. Delta hedging as usual, our portfolio is $\Pi = V - \Delta S$, and the instantaneous change in Π is

$$\begin{aligned}\mathrm{d}\Pi =& \Big(V_t + \frac{1}{2}\sigma^2 V_{SS}\Big)\mathrm{d}t + (\upsilon S \mathrm{d}t + \sigma S \mathrm{d}W)V_S - \sigma(\Lambda - \nu)\frac{\partial V}{\partial \Lambda}\mathrm{d}t \\ &- \Delta(\upsilon S \mathrm{d}t + \sigma S \mathrm{d}W) \\ &+ \mathrm{d}N_t\Big[V(YS, t, \Lambda + \alpha) - V(S, t, \Lambda)\Big] - \Delta(YS - S)\end{aligned}$$

where, as previously, υ is the drift of the underlying and subscripts mean partial differentials (except for N_t, the counting process). By no arbitrage arguments, we may assume that $\mathbb{E}(\mathrm{d}\Pi) = r\Pi \mathrm{d}t$ and therefore,

$$\begin{aligned}0 =& V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + (r - \lambda\kappa)SV_S - rV - \beta(\lambda - \nu)\frac{\partial V}{\partial \Lambda} \\ &+ \lambda\bigg(\int_0^\infty V(YS, t, \Lambda + \alpha)g(Y)\mathrm{d}Y - V(S, t, \Lambda)\bigg)\end{aligned}$$

where $g(\cdot)$ is the jump amplitude distribution.

A.3.7 Distribution of N_t for Hawkes Hedging Experiments

In Table A.1, we give $\mathbb{P}(N_t = n)$, which are used for experiments in section 4.2.4.

$\mathbf{N_t}$	**Hawkes**	**Poisson**
0	0.989807	0.975310
1	0.005310	0.024383
2	0.002075	0.000305
3	0.000854	0.000003
4	0.000488	0.000000
5	0.000488	0.000000
6	0.000244	0.000000
7	0.000183	0.000000
8	0.000122	0.000000
9	0.000122	0.000000
10	0.000122	0.000000
11	0.000122	0.000000
12	0.000061	0.000000

Table A.1: Hawkes probabilities compared to Poisson probabilities for each value of N_t up to 12 (to 6 decimal places). $\lambda = 0.1$ and for Hawkes, $\nu = 0.01$, $\beta = 10$. The exact values of each column sum to 1.

The distribution is truncated at $N_t = 12$, which is reasonable considering $\mathbb{E}(N_t) = \lambda u = 0.1 * 0.25 = 0.025$. A Hawkes process is simulated using $N = 2^8$ time intervals and $M = 2^14 = 16,384$ paths, with the resulting probabilities in Table A.1. Due to the limititations of the Monte Carlo estimation, these probabilities are accurate to less than the 6 decimal places shown.

Appendix B

MATLAB Code

B.1 Valuation & the Greeks

B.1.1 European Options under Black-Scholes

```
% Black Scholes call or put option price and Greeks

% As defined in equation (3.17) on page 48 of
% The Mathematics of Financial Derivatives
% by Wilmott, Howison and Dewynne

% Original code by Mike Giles
% Adapted by Nick Hinde

function V = eBS(r,sigma,T,S,K,opt,type)

    % Default type == 'call'
    if nargin == 6
      type = 'call';
    elseif nargin ~= 7
      error('wrong number of arguments');
    end

    if sum(S <= 0) > 0
        error('S has nonpositive elements');
    end

    d1 = ( log(S./K) + (r+0.5*sigma.^2).*T )./ (sigma.*sqrt(T));
    d2 = ( log(S./K) + (r-0.5*sigma.^2).*T ) ./ (sigma.*sqrt(T));

    % Note: gamma and vega are identical for call and put

    switch opt
      case 'value'
          switch type
              case 'call'
                V =  S.*N( d1) - K.*exp(-r.*T).*N( d2);
```

```
            case 'put'
              V = -S.*N(-d1) + K.*exp(-r.*T).*N(-d2);

            case 'straddle'
              V = eBS(r,sigma,T,S,K,opt,'call') + eBS(r,sigma,T,S,K,opt,'put');

            otherwise
                error('invalid value for type -- must be ''call'' or ''put''')
        end

    case 'delta'
        switch type
            case 'call'
                V = N(d1);

            case 'put'
                V = N(d1) - 1;

            case 'straddle'
                V = eBS(r,sigma,T,S,K,opt,'call') + eBS(r,sigma,T,S,K,opt,'put');
        end

    case 'gamma'
      V = exp(-0.5*d1.^2) ./ (sigma*sqrt(2*pi*T).*S);

    case 'vega'
      V =        S.*(exp(-0.5*d1.^2)./sqrt(2*pi)).*( sqrt(T)-d1/sigma) ...
        - exp(-r*T)*(exp(-0.5*d2.^2)./sqrt(2*pi)).*(-sqrt(T)-d2/sigma);

    otherwise
      warning('invalid value for opt -- must be ''value'', ''delta'' or ''gamma''')
      V = zeros(length(S));
  end

end % function


function cum = N(x)
% cumulative normal distribution function

    xr = real(x);
    xi = imag(x);

    if abs(xi)>1e-10
      error 'imag(x) too large in N(x)'
    end

    cum = 0.5*(1+erf(xr/sqrt(2))) + i*xi.*exp(-0.5*xr.^2)/sqrt(2*pi);

end % function
```

B.1.2 European Options under Merton's Jump-Diffusion

```
% Merton's jump-diffusion call and put option price and Greeks

% European call or put based on Merton model of Brownian motion
% plus jumps with a Poisson rate and log-Normal size
%
% Assume strike K = 1
%
% lambda- jump intensity
% mu    - log of jump mean
% del   - variability of jump
% r     - interest rate
% sigma - volatility
% T     - time interval
% S     - asset value(s)
% K     - strike
% opt   - 'value', 'delta', 'gamma', 'vega', 'lambda', 'mu', 'del'
% type  - 'call' or 'put'
% V     - option value(s)

% Original code by Mike Giles
% Adapted by Nick Hinde

function V = eMerton(lambda,mu,del,r,sigma,T,S,K,opt,type)

    if nargin ~= 10
      error('wrong number of arguments');
    end

    % For the purposes of this application, this N gives sufficient accuracy
    % Error in truncating the series solution
    %   eMerton(0.1,-0.9,0.4,0.05,0.2,1,1,1,'delta','put',3)...
    % - eMerton(0.1,-0.9,0.4,0.05,0.2,1,1,1,'delta','put',50)
    % This is so small that we need not compute more than 25*lambda*T
    % The relative error for our parameters is 1e-3 or less
    N = ceil(25*lambda*T);

    tol = 1e-8;

    % Ej = log( expectation of jump amplitude )
    Ej = mu+0.5*del^2;

    V = 0;

    % n is the number of jumps
    for n = 0:N
      if T ~= 0
          sig_n = sqrt(sigma^2 + n*del^2/T);
          S0_n  = S*exp( (1-exp(Ej))*lambda*T + n*Ej);

          if n==0
            prob = exp(-lambda*T);
          elseif lambda==0
```

```
          prob = 0;
        else
          prob = exp(-lambda*T + n*log(lambda*T) - sum(log(1:n)));
        end
    end

    switch opt
      case 'value'
          if abs(T) < tol
              % Note it is not necessary for this line to run N times
              % so prematurely end the loop n = 0:N
              V = payoff(S,K,type);
              n = N;
          else
              val = eBS(r,sig_n,T,S0_n,K,opt,type);
              V = V + prob.*val;
          end
      case 'delta'
        val = eBS(r,sig_n,T,S0_n,K,opt,type);
        V = V + prob*val.*S0_n./S;
      case 'gamma'
        val = eBS(r,sig_n,T,S0_n,K,opt,type);
        V = V + prob*val.*(S0_n./S).^2;
      case 'vega'
        val = eBS(r,sig_n,T,S0_n,K,opt,type);
        V = V + prob*val.*sigma/sig_n;

      case 'lambda'
        if n==1
          prob2 = exp(-lambda*T);
        elseif lambda==0 | n==0
          prob2 = 0;
        else
          prob2 = exp(-lambda*T + (n-1)*log(lambda*T) - sum(log(1:n-1)));
        end
        val = eBS(r,sig_n,T,S0_n,K,'value',type);
        V = V + T*(prob2-prob)*val;
        val = eBS(r,sig_n,T,S0_n,K,'delta',type);
        V = V + prob*val.*S0_n*(1-exp(Ej))*T;

      case 'mu'
        val = eBS(r,sig_n,T,S0_n,K,'delta',type);
        V = V + prob*val.*S0_n*(-exp(Ej)*lambda*T + n);
      case 'del'
        val = eBS(r,sig_n,T,S0_n,K,'vega',type);
        V = V + prob*val*n*del/sig_n;
      otherwise
      error('invalid value for opt -- must be ''value'', ''delta'',...
            ''gamma'', ''vega'', ''lambda'', ''Ej'' or ''del''')
    end
  end

end % function
```

B.1.3 Normal Distribution Function & its Derivatives

```
% Evaluate the Normal distribution
% and its derivatives
%
% output: m^th derivative of standard Normal distribution
%
% By Mike Giles

function y = Normal(x,m)

    if nargin == 1
      m = 0;
    end

    if m == 0
      y = (1/sqrt(2*pi))*exp(-x.^2/2);
    elseif m == 1
      y = (1/sqrt(2*pi))*(-x).*exp(-x.^2/2);
    elseif m == 2
      y = (1/sqrt(2*pi))*(-1+x.^2).*exp(-x.^2/2);
    elseif m == 3
      y = (1/sqrt(2*pi))*(3*x-x.^3).*exp(-x.^2/2);
    elseif m == 4
      y = (1/sqrt(2*pi))*(3-6*x.^2+x.^4).*exp(-x.^2/2);
    end

end % function
```

B.1.4 Comparing Black-Scholes and Merton Value & Greeks

```
% Plot graphs of Merton vs Black-Scholes (both analytic)

function [] = plotMertonVsBS()

    clear all
    close all;

    % Underlying parameters
    r     = 0.05;
    sigma = 0.2;

    % Option parameters
    T = 3;
    K = 1;
    type = 'put';

    opt = 'value';

    % Jump parameters
```

```
% Edited by Nick
lambda = 0.1; % lam_j = 0.1;
mu  = -0.92;
del =  0.425;

% initial spot (1 means 100% of spot)
Smin = 0.0001;
Smax = 3;
NS = 512;
S = linspace(Smin, Smax, NS+1);

% Compensated drift and volatility
% kappa = exp(mu + 0.5*del^2)-1;
% rC = r-lambda*mu;
% sigmaC = sqrt(sigma^2+lambda*del^2);

figure;

for i = 1:1:8

    if((i/2) == ceil(i/2))
        type = 'call';
    else
        type = 'put';
    end

    if(i == 1 | i == 2)
        opt = 'value';
    elseif (i == 3 | i == 4)
        opt = 'delta';
    elseif (i == 5 | i == 6)
        opt = 'gamma';
    elseif( i == 7 | i == 8)
        opt = 'vega';
    end

    windowPanes(4,2,i,0.05,0.03);
    grid on;

    VM  = eMerton(lambda,mu,del,r,sigma,T,S,K,opt,type);
    VBS = eBS(r,sigma,T,S,K,opt,type);

    hold on;
    plot(S,VM, 'k--', 'LineWidth', 2);
    plot(S,VBS,'k-');
    if(i < 3)
        axis equal;
        axis([0 Smax 0 2])
    elseif i == 3 | i == 4
        axis equal
        if i == 3
            axis([0 Smax -1 1]);
        else
            axis([0 Smax -1 1]);
```

```
            end
        end
        %legend('Merton', 'Black-Scholes');
        xlabel('S', 'FontSize', 14);
        if (((i+1)/2) == ceil((i+1)/2))
            ylabel(opt, 'FontSize', 14);
    end

end % function
```

B.2 Monte Carlo Estimation

B.2.1 Creating Sample Asset Price Paths under Jump-Diffusion

```
% Create jump-diffusion asset price paths

% Evolution:  dS = r*S dt + sigma*S dW + jumps
%
% X = log(S)
%
% lambda- intensity
% mu    - jump amplitude mean
% del   - variance of jump amplitude
% r     - interest rate
% sigma - volatility
% S0    - initial asset price
% T     - time interval
% M     - number of paths
% N     - time intervals
% S     - terminal asset price(s)

function [t X] = jumpDiffusionSpot(lambda, mu, del, r, sigma, T, S0, M, N)

    % Maximum number of jumps in [0,T] allowed (approximation to exact)
    Mj  = ceil(3*real(lambda)*T);

    % construct jump times and amplitudes

    pj = rand(M,Mj);
    tj = cumsum( (-log(pj))/lambda, 2);
    sj = mu + del*randn(M,Mj);

    t  = linspace(0,T,N+1);

    % Set up paths
    X = zeros(M,N+1);

    % loop over timesteps
    h = T/N;
```

```
    X(:,1) = log(S0);

    kappa = exp(mu+0.5*del^2)-1;
    rm  = r - 0.5*sigma^2 - lambda*kappa;

    sigmaRootH = sigma*sqrt(h);
    rmH = rm*h;

    Xnow = X(:,1);

    for n = 2:N+1
      sigmaRootHdW = sigmaRootH * randn(M,1);

      Xnow = Xnow + rmH + sigmaRootHdW;
      X(:,n)  = Xnow;
    end

    % Sharp jumps
    for n = 2:N+1
       X(:,n) = X(:,n) + sum( (tj<t(n)).*sj, 2);
    end

end % function
```

B.2.2 Monte Carlo Estimation of Value under Jump-Diffusion

```
% The price of a European option using Monte Carlo

% Assume strike K = 1

% r - interest rate
% T - time to expiry
% S - simulations of the terminal asset price
% type - option contract: 'put' or 'call'
% M - number of simulations

function [VMC stdev] = monteCarlo(lambda,mu,del,r,sigma,T,S,M,Nt,opt,type)

    VMC   = zeros(size(S));
    stdev = zeros(size(S));

    S0 = 1;

    % simulations
    [t X] = jumpDiffusionSpot(lambda,mu,del,r,sigma,T,S0,M,Nt);
    size(X)
    ST = exp( X(:,Nt+1) );

    for k = 1:1:length(S)
```

```
        terminal  = ST*S(k);

        % discounted payoff
        switch type
            case 'call'
                V = exp(-r*T)*max(terminal - 1, 0);
            case 'put'
                V = exp(-r*T)*max(1 - terminal, 0);
        end

        VMC(k) = sum(V)/M;
        stdev(k) = sqrt( (sum(V.^2)/M - VMC(k)^2)/(M-1) );

    end

end % function
```

B.2.3 Plots of the Monte Carlo Error & Standard Deviation

```
% Test of Monte-Carlo results against Merton's analytical formulae

function [] = testMC()

    clear all; close all;

    % Underlying parameters
    r     = 0.05;
    sigma = 0.2;

    % Option parameters
    T = 1;
    K = 1;
    type = 'call';

    opt = 'value';

    % Jump parameters
    lambda = 0.1;
    mu = -0.92;
    del= 0.425;

    % Number of time intervals
    Nt = 64; % 256;

    % initial spot (1 means 100% of spot)
    Smin = 0.5;
    Smax = 1.5;
    NS = 64;
    S = linspace(Smin, Smax, NS+1);
```

```
    % Number of terms in series solution
    N = 100;

    figure;

    for j = 1:1:2

        if j == 1
            type = 'put';
        else
            type = 'call';
        end

        VM  = eMerton(lambda,mu,del,r,sigma,T,S,K,opt,type,N);
        VBS = eBS(r,sigma,T,S,K,opt,type);

        windowPanes(1, 2, j, 0.03, 0.1);

        % Number of simulations
        M   = ceil(exp(linspace(log(10),log(5e4),30)));

        for i = 1:1:length(M)

            [VMC sigmaMC] =...
            monteCarlo(lambda,mu,del,r,sigma,T,S,M(i),Nt,opt,type);

            error(i) = norm(VMC - VM)/sqrt(NS+1);
         relError(i) = norm((VMC - VM)./VM)/sqrt(NS+1);
            stdev(i) = sum(sigmaMC)/length(sigmaMC);
        end

        loglog(M,stdev, 'k.', 'MarkerSize', 15);
        hold on;
        M1 = M(1);
        M2 = M(length(M));
        stdev2 = stdev(length(stdev));
        loglog([M1 M2], [sqrt(M2/M1)*stdev2 stdev2], 'k-');
        xlabel('M', 'FontSize', 14);
        ylabel('Standard Deviation', 'FontSize', 14);
        axis([-inf inf -inf inf]);

    end

end % function
```

B.2.4 Simulation of Hawkes Processes

```
% Simulating a Hawkes process
% with exponential decay as the integral weight

function [] = Hawkes()

    clear all;
    close all;

    global lambda nu alpha beta tmid k

    % Mean intensity (rate) = E(Lambda)
    lambda = 0.1;

    % Intensity if there have been no jumps
    nu = 0.005; % [0.04  0.06 0.08 0.10]

    % Memory decay
    beta = 1;

    % Importance of the cumulative sum of previous jumps
    alpha = beta*(1-nu/lambda);

    % As a check, these should be equal:
    % lambda
    % nu * beta / (beta - alpha)

    % Simulations
    M = 2^10 % 2048
    Mplot = 64;

    % Time intervals
    Nt = 128;

    % Time frame of the simulation
    T = 30;
    t = linspace(0,T,Nt+1);
    dt = t(2)-t(1);
    tmid = linspace(dt,T-dt,Nt);

    lambdaT = lambda*T
    plotPoisson(nu*10*lambdaT);
    hold on;

    for k = 1:1:length(nu)

        % Plot the actual simulations (up to Mplot)
        % plot(1:1:Mplot,t_j(1:Mplot,:), 'k.');
        % xlabel('Simulation', 'FontSize', 14);
        % ylabel('Time', 'FontSize', 14);
        % axis([1 Mplot 0 T]);
        % hold on;
        %
```

```
        % for m = 1:1:Mplot
        %     plot([m m], [0 T], 'k-');
        % end

        t_j = HawkesPath(k,M,Nt,t,tmid,dt);

        events = sum(t_j>0,2);
        sum(events)/M

        % Plot distribution of number of events
        min = -0.5;
        max = 3*lambdaT+0.5;
        [emid density] = distribution(events,min,max,ceil(max-min));
        if k == 1
            plot(emid, density, '*k');
        elseif k == 2
            plot(emid, density, '^k');
        elseif k == 3
            plot(emid, density, 'squarek');
        elseif k == 4
            plot(emid, density, 'ok');
        elseif k == 5
            plot(emid, density, '.k');
        elseif k == 6
            plot(emid, density, 'diamondk');
         end

    end

    axis([min max 0 inf]);
    xlabel('Events', 'FontSize', 14);
    ylabel('Frequency Density', 'FontSize', 14);

    legend('Poisson',...
        strcat('Simulation: \nu=', num2str(nu(1))));

end % function


function [t_j] = HawkesPath(k,M,Nt,t,tmid,dt)
% k is the index of nu
% M is the number of paths

    % jumps (1 if a jump occurs, 0 otherwise)
    % To ensure that all the elements of the matrix are used
    % let there be lots of jumps a very long way in the past
    t_j = -1e20*ones(M,Nt);

    for i = 1:1:Nt
        Lambda = intensity(t(i), t_j);

        % Jump times: assume jump happened at the midpoint of the interval
        t_j( rand(M,1) <= Lambda*dt, i ) = tmid(i);
    end
```

```
end % function


function [Lambda] = intensity(t, t_j);
% t is a scalar
% t_j is a vector of jump times

    global lambda nu alpha beta tmid k

    % The last term on this line is very computationally expensive
    Lambda = nu(k) + (lambda-nu(k))*exp(-beta*t)...
             + alpha(k)*sum(exp( -beta*(t-t_j) ), 2);

end % function


function [xmid density] = distribution(x,xmin,xmax,N)
% Plot the final distribution

    bins = linspace(xmin, xmax, N+1);
    dx = bins(2)-bins(1);

    frequency = zeros(N,1);
    for i = 1:1:N
        frequency(i) = sum( bins(i) <= x & x < bins(i+1) );
    end

    xmid = linspace(bins(1)+dx/2, bins(N+1)-dx/2, N);

    density = frequency/length(x);

end % function


function [] = plotPoisson(mean);

    N = 5*mean;

    n = 0:1:N;

    for i = 0:1:N
        p(i+1) = mean^ i / factorial(i) * exp(-mean);
    end

    bar(n,p)
    colormap([1 1 1]);

end % function
```

B.2.5 Generating Sample Terminal Asset Prices for Hawkes

```
% Obtain jump-diffusion terminal asset price

% Using a Hawkes process for arrivals

function [t X] = terminalSpot(lambda, mu, del, r, sigma, T, S0, M, N)

    t  = linspace(0,T,N+1);

    % Set up paths
    X = zeros(M,N+1);

    % loop over timesteps
    h = T/N;
    X(:,1) = log(S0);

    kappa = exp(mu+0.5*del^2)-1;
    rm  = r - 0.5*sigma^2 - lambda*kappa;

    sigmaRootH = sigma*sqrt(h);
    rmH = rm*h;

    Xnow = X(:,1);

    for n = 2:N+1
      sigmaRootHdW = sigmaRootH * randn(M,1);

      Xnow = Xnow + rmH + sigmaRootHdW;
      X(:,n)  = Xnow;
    end

    % HAWKES JUMPS

    % Uses nu = 0.01 and beta = 10
    P=[0.98980712890625
       0.00531005859375
       0.00207519531250
       0.00085449218750
       0.00048828125000
       0.00048828125000
       0.00024414062500
       0.00018310546875
       0.00012207031250
       0.00012207031250
       0.00012207031250
       0.00012207031250];

    pj = ones(M,1)*cumsum(P');
    Mj = length(P);

    % The last probability is 0.00006103515625
    % Then the probabilities do indeed sum to 1.
```

```
    % Maximum number of jumps in [0,T] allowed (approximation to exact)
    % THIS IS ONLY APPROPRIATE FOR U = 0.25 AND LAMBDA = 0.1

    % construct jump amplitudes
    sj = mu + del*randn(M,Mj);

    % Uniform random numbers
    ran = rand(M,1)*ones(1,Mj);

    % Add jumps
    X(:,N+1) = X(:,N+1) + sum((ran>pj).*sj,2);

end % function
```

B.3 Numerically Solving the PIDE

B.3.1 Valuing American Options using the PIDE

```
% The price of an American put option under jump-diffusion
% By Alex Prideaux and Nick Hinde

% This uses extensively the paper:
% A Penalty Method for American Options with Jump Diffusion Processes
% (Forsyth, Halluin, Labahn)

% S  - asset prices
% V  - Value of American option
% Ve - Value of European option
% Delta - dC/dS for the American option
% E  - exercise boundary

function [S t Vamerican Veuropean delta E]...
= aPIDE(lambda,mu,del,r,sigma,T,K,type, N, Nt, dynamic, xmin, xmax);
% Space discretization is uniform in log(S)

    % theta = 0 is implicit, theta = 1 is explicit

    % for differential operator FDS
    theta   = 0.5;
    % for the integral operator
    theta_j = 0.5;

    % Make the intervals the same size for x and ex
    x  = linspace(xmin, xmax, N+1);
    ex = linspace(2*xmin-xmax,2*xmax-xmin, 3*N+1);
    S  = exp(x);
    eS = exp(ex);
    dx = (xmax-xmin)/N;
```

```
% Initial timestep
dt = T/Nt;

sig2 = sigma^2;
k = exp(mu + (del^2)/2) - 1;

% Set up FDS matrix entries - using the theta scheme
a = 1/(2*dx) * (sig2/dx   - r + lambda*k + sig2/2);
b =          - (sig2/dx^2 + r + lambda);
c = 1/(2*dx) * (sig2/dx   + r - lambda*k - sig2/2);

% spdiags is very slow, for example:
% n = 1000; a= 2; tic, A =spdiags(a,[0], n+1,n+1); toc;
% tic; sparse(1:n+1,1:n+1,2*ones(1,n+1),n+1,n+1); toc;
% A = spdiags([[a,0, 0]', [0 b 0]', [0 0 c]'],[-1 0 1],N+1,N+1);
A =   sparse(2:N,2:N,  b*ones(1,N-1),N+1,N+1)...
    + sparse(2:N,1:N-1,a*ones(1,N-1),N+1,N+1)...
    + sparse(2:N,3:N+1,c*ones(1,N-1),N+1,N+1);
I = speye(N+1,N+1);

% Construct a vector g for the integral
gauss  = @(y) (exp(-((y-mu).^2)/(2*del^2))/(sqrt(2*pi)*del));
% Turn gauss into a vector function corresponding to the domain x
ef = @(y) (gauss(y + x.'));
g = lambda*quadv(ef, -dx/2, dx/2, 1e-10);
g = g.';

% Create G
col = zeros(1,N+1);
col(1) = g(1);
row = zeros(1,2*N+1);
row(1:N+1) = g;
G = toeplitz(col,row);

G1 = G(:,      1:N/2);
G2 = G(:,  1+N/2:N/2*3+1);
G3 = G(:,2+3*N/2:N*2+1);

B = G2;
% If mu = 0, B should be symmetric.  Check symmetry:
% spy(abs(B-B.')>1e-16)
B_i = (1-theta_j)*B; % implicit
B_e =     theta_j*B; % explicit

% Terminal condition
VT  = payoff(S, K,type).';
eVT = payoff(eS,K,type).';

% Extended payoff - this could be replaced by pure diffusion solution
VT1 = eVT(1+N/2:N);
VT3 = eVT(2*N+2:5*N/2+1);

% Constant vector for either end of the integral
h = G1*VT1 + G3*VT3;
```

```
% Change to tol (was 1e-12)
tol = 1e-8;
kmax = 20;
large = min(1e4, 1/tol);

V  = VT;
Ve = VT;
Vold = zeros(N+1);

E = zeros(Nt+1,1);
switch type
    case 'put'
        E(1) = S(N/2);
end
integers = 1:N+1;

t = T;
i = 1;

% Store values and boundary
timeIndex = Nt+1;
Veuropean(i,:) = Ve;
Vamerican(i,:) = V;

i = i+1;


while dt*1e-2 < abs(t)

    % Use dynamic time-stepping if dynamic == 1
    if(dynamic == 1)
        dt = timestep(dt, t, T, Vold, V, N);
    end % if

    if(t-dt < dt*1e-2)
        dt = t;
    end % if

    [LHS LHSa RHS] = differentialOperators(I, A, N, theta, dt, r, lambda);

    amer_exp = RHS*V  + dt*(B_e*V + h);
    euro_exp = RHS*Ve + dt*(B_e*Ve + exp(-r*(T-t))*h);

    diff = tol*2;

    Vk = V;
    k = 1;
    while diff > tol && k <= kmax
        % Penalty vector
        P = zeros(N+1,1);
        P( Vk < VT ) = large;

        % old code: spdiags(P,0,N+1,N+1);
```

```
        LHSandP = LHSa + sparse(1:N+1,1:N+1,P,N+1,N+1);

        % THE MOST EXPENSIVE OPERATION IS B_i*Vk
        denseResult = B_i*Vk;
        RHSvector = amer_exp + dt*denseResult + P.*VT;
        % Tridiagonal solve
        Vnew = LHSandP \ RHSvector;

        diff = max( abs(Vnew - Vk) ./ max(1, real(Vnew)) );

        Vk = Vnew;
        k = k+1;
    end % while

    % Obtain exercise boundary
    change = P(2:N+1)-P(1:N) ~= 0;
    change(1:ceil(N/3)) = 0;
    index = integers(change);
    if size(index) ~= [0 0]
        E(i) = (S(index)+S(index+1))/2;
    end

    % Instead of doing actual implicit, which involves solving
    %          a linear system, do faked one
    Ve1 = LHS \ (euro_exp + dt*(B_i*Ve ));
    Ve2 = LHS \ (euro_exp + dt*(B_i*Ve1));
    Ve  = LHS \ (euro_exp + dt*(B_i*Ve2));

    Vold = V;
    V = Vk;

    % Store values and boundary
    Veuropean(i,:) = Ve;
    Vamerican(i,:) = V;

    t = t-dt;
    i = i+1;

end; % while


% Compute delta
j = 2:1:N;
delta = zeros(Nt+1,N+1);
dS = (S(j+1)-S(j-1)) / 2;
for i = 1:1:Nt+1
    % Use central divided difference
    delta(i,:) = [-1 ( Vamerican(i,j+1)-Vamerican(i,j-1) )./(2*dS) 0];
end

delta(:,1) = delta(:,2);

% Store times
t = linspace(0,T,Nt+1);
```

```
end % function


function [LHS LHSa RHS] = differentialOperators(I, A, N, theta, dt, r, lambda)
% I and A should be sparse (A tridiagonal)

    LHS = I - (1-theta) * dt * A;
    RHS = I +    theta  * dt * A;

    % European
    LHS(1,1) = 1+(r+lambda)*dt;
    LHS(N+1,N+1) = 1;

    % American
    LHSa = LHS;
    LHSa(1,1) = 1+lambda*dt;
    LHSa(N+1,N+1) = 1;

    % Both European and American
    RHS(1,1) = 1;
    RHS(N+1, N+1) = 1;

end % function


% VARYING THE TIMESTEP dt according to
% dtnew = { min_i [dnorm / |V(S_i,t-dt) - V(S_i,t)|
%                 / max(D, |V(S_i,t-dt)|, |V(S_i,t)| ] } * dt
% where time discretization is t, t-dt, t-dt-dtnew

function [dt] = timestep(dt, t, T, Vold, V, N);

    D = 1.0;
    dnorm = 0.05;

    V    = real(V);
    Vold = real(Vold);

    if(t~=T)
        % dtnew = dt * { min_i [dnorm / |V(S_i,t-dt) - V(S_i,t)|...
                               / max(D, |V(S_i,t-dt)|, |V(S_i,t)| ] }
        denom = max( max( D, abs(V) ), abs(Vold) );

        warning off % for divide by zero
        dt = dt * min( dnorm ./ abs(V-Vold) ./ denom )
        warning on
    end % if

end % function
```

B.3.2 The Error as a Function of the Grid Fineness

```
% Test PIDE solver for errors as a function of N
% where N is the number of intervals in x

function [] = errorWithN()

    clear all, clc
    format compact

    r      = 0.05;
    sigma  = 0.15;
    T      = 0.25;
    del    = 0.45;
    mu     =-0.90;
    lambda = 0.10;
    opt    = 'value';
    type   = 'put';

    % Used stored values instead of calculating actuals
    load ('american_value.mat','Sa','VBSa');

    dt = 0.002;

    N = 100:200:1500;

    for j=2:1:2

        if(j==1)
            % dynamic timestepping
            dynamic = 1;
        else % j~=1
            % fixed timestepping
            dynamic = 0;
        end

        for i=1:1:length(N)

            N(i)

            [S Vj Vje] = aPIDE(lambda,mu,del,r,sigma,T,opt,type, N(i),dt,dynamic);
            [S Va Ve]  = aPIDE(     0, 1,  1,r,sigma,T,opt,type, N(i),dt,dynamic);

            Vm      = e_merton(lambda,mu,del,r,sigma,T,S,opt,50,type);
            VBS     = e_vanilla(r,sigma,T,S,opt,type);

            VBSaSpline = spline(Sa', VBSa', S);

            denom = sqrt(length(S));

            % Error = ||exact - approx||_2
            errorBSe(i) = norm(Ve'  - VBS       )/denom;
            errorBSa(i) = norm(Va'  - VBSaSpline)/denom;
            errorMj(i)  = norm(Vje' - Vm        )/denom;
```

```
        end

        figure;
        loglog(N, errorBSe, 'k-');
        hold on;
        loglog(N, errorBSa, 'k-.');
        loglog(N, errorMj, 'k--');

        legend('Black-Scholes European', 'Black-Scholes American',...
               'Merton European');
        xlabel('N', 'FontSize', 12);
        ylabel('Error = ||exact - approx||_2 / sqrt(no.elements)',...
               'FontSize', 12);
        axis([N(1) N(length(N)) -inf inf]);
        hold off;

        if(dynamic == 1)
            title('Dynamic time-stepping', 'FontSize', 12);
        else % dynamic ~= 1
            title(strcat('Fixed time-stepping, dt = [', num2str(dt),...
                         '] T = 0.25'), 'FontSize', 12);
        end

    end

end % function
```

B.3.3 The Error as a Function of Asset Price S

```
% Plot the relative error in the numerical solution to the PIDE

function [] = plotError();

    close all
    clear
    format compact
    clc

    r      = 0.05;
    sigma  = 0.15;
    T      = 0.25;
    del    = 0.45;
    mu     =-0.90;
    lambda = 0.10;
    opt    = 'value';
    type   = 'put';

    Nx = 1500;
    Nt = 30;
```

```
dynamic = 0;
xmin = -6;
xmax = 6;

for i = 1:1:2

    if i==1
        type = 'put'
    else
        type = 'call'
    end

    windowPanes(2, 1, i, 0.05, 0.05);

    [S Vj Vje E] = aPIDE(lambda,mu,del,r,sigma,...
                         T,opt,type,Nx,Nt,dynamic,xmin,xmax);
    [S Va Ve  E] = aPIDE(     0, 1,  1,r,sigma,...
                         T,opt,type,Nx,Nt,dynamic,xmin,xmax);

    Vm  = e_merton(lambda,mu,del,r,sigma,T,S,opt,50,type);
    VBS = e_vanilla(r,sigma,T,S,opt,type);

    Vm = Vm';
    VBS = VBS';

    S = S';

    switch type
        case 'put'
            load ('american_value.mat','Sa','VBSa');
            VBSa = spline(Sa,VBSa,S);
        case 'call'
            Sa = S;
            VBSa = VBS;
    end

    B = Vm > 0 & Vje > 0;
    Vm = Vm(B);
    Vje = Vje(B);
    Se = S(B);

    sum(Vm == 0)
    sum((Vje-Vm) == 0)

    % PLOT RELATIVE ERROR
    hold off;

    semilogy(Se, abs((Vje-Vm)./Vm), 'k-.');
    hold on;
    semilogy(S, abs((Va-VBSa)./VBSa), 'k-');
    semilogy(S, abs((Ve-VBS)./VBS), 'k--', 'LineWidth', 2);

    legend('Merton European', 'Black-Scholes American',...
```

```
            'Black-Scholes European');
        xlabel('S', 'FontSize', 14);
        ylabel('Relative Error', 'FontSize', 14);

        if i == 1
            axis([0 2 1e-9 1]);
        else
            axis([0 2 1e-9 1]);
        end

    end % for

end % function
```

B.4 Strategies

B.4.1 Delta Hedging

```
% Merton's delta hedging strategy - use Merton's formula for delta

function [Kh Wh WS WB] = deltaHedge(N,t,dt,St,Ht,Sdelta,delta)
% Note: N, t and dt are not used by this function

    NS = length(Sdelta)-1;
    Smin = Sdelta(1);
    Smax = Sdelta(NS+1);
    Sdomain = Smax-Smin;

    dS = Sdomain/NS;
    lower = Smin/dS;

    % Find nearest index
    i = 1 + round( St/dS - lower );

    % extrapolation outside [Smin Smax]
    i(i > NS+1) = NS+1;
    i(i < 1) = 1;

    WS = delta(i);

    % Check error is small
    % sum(WS1-WS)/length(WS)

    WB = Ht - WS.*St;
    Kh = 0;
    Wh = 0;

end % function
```

B.4.2 Least Squares Solution for Hedging Option Weights

```
% Least squares hedging, given an integral weighting function

% N is the number of hedging instruments (excluding underlying asset)
% Target option exprires at T
% Hedging options expire at u
% Time now is t
% Time period over which to hedge is dt
% Asset price now is St (which may be a vector)
% Value of the hedge at time t is Ht

function [Khedge Wh WS WB] = leastSquares(N,t,dt,St,Ht,integralWeighting)

    global r sigma S0 lambda mu del T K ttype htype u M tN Kh

    % Note this could be 2^8 for greater accuracy, but is slower
    wN = 2^6;

    % Note these nodes are in terms of x = log(S)
    [nodes w] = integralWeighting(wN,dt);

    Khedge = Kh(1:N);

    for j = 1:1:length(St)

        Send = St(j)*nodes;

        % It and Iend are the hedging instruments' value at t and t+dt
        % It and Iend should be wN by N

        It = eMerton(lambda,mu,del,r,sigma,u-t,St(j),Khedge,'value',htype);
        It(N+1) = St(j);
        It = ones(wN,1)*It;

        Iend = zeros(wN,N+1);
        for i=1:N
            Iend(:,i) = eMerton(lambda,mu,del,r,sigma,u-t-dt,...
                                Send,Khedge(i),'value',htype);
        end
        Iend(:,N+1) = Send;

        % To ensure it is not rank-deficient,
        % there must be nodes > (<) all strikes Khedge
        % rank(Iend)

        Vend = eMerton(lambda,mu,del,r,sigma,T-t-dt,Send,K,'value',ttype);

        % Weighted least squares: (A*phi - b)_i^2.w_i
        % where A is a matrix, b and w are vectors
        b = Ht(j)*exp(r*dt) - Vend;
        b = b';
        A = It*exp(r*dt) - Iend;
```

```
        % If the weighting function is the jump amplitude PDF
        % impose delta neutrality at time t
        if strcmp( func2str(integralWeighting), 'jumpPDFapprox' ) == 1

            hedgeValue = eMerton(lambda,mu,del,r,sigma,u-t,...
                                 St(j),Khedge,'delta',htype);
            Idelta(1,1:N) = hedgeValue.';
            Idelta(1,N+1) = 1;
            Vdelta        = eMerton(lambda,mu,del,r,sigma,T-t,...
                                    St(j),K, 'delta',ttype);

            A(wN+1,:) = Idelta;
            b(wN+1)   = Vdelta;

            % Place emphasis on delta neutrality
            w(wN+1) = 1e2;

            % Normalize w
            w = w/sum(w);
        end

        % phi is the weight of each instrument in the hedge
        warning off;
        phi = lscov(A,b,w);
        warning on;

        % The weighted inner product should be minimized by phi
        % Check that the result is very small
        % F = A*phi-b;
        % (F'*diag(w))*(F.*w)

        Wh(j,:) = phi(1:N);
        WS(j,1) = phi(N+1);

    end

    Khedge = ones(length(St),1)*Khedge;

    % Cost of hedge excluding bonds at time t
    cost = evaluate(St.',t-dt,dt,Khedge,Wh,WS,0);

    % Bond weight = hedging funds - cost of hedging options and asset
    WB(:,1) = Ht.' - cost;

end % function
```

B.4.3 Least Squares Weighted by the Transition PDF

```
% Hedge so as to minimize the quadratic error
% weighted by the transition PDF

% This is usually used as a semi-static strategy

% N is the number of hedging instruments (excluding underlying asset)
% Target option exprires at T
% Hedging options expire at u
% Time now is t
% Time period over which to hedge is dt
% Asset price now is St

function [Kh Wh WS WB] = leastSquaresError(N,t,dt,St,Ht,Sdelta,delta)
% Note: Sdelta and delta are not used by this function

    disp(sprintf('Error Strategy'));

    [Kh Wh WS WB] = leastSquares(N,t,dt,St,Ht,@transitionPDFapprox);

end % function


function [S w] = transitionPDFapprox(wN,dt)
% Create nodes and weights for approximating the transition PDF
% wN is the number of nodes

    global r sigma S0 lambda mu del T K u M tN

    tol = 1e-3;

    xmin = -1e-3;
    while PDF(xmin,dt) > tol
        xmin = 1.2*xmin;
    end

    xmax = 1e-3;
    while PDF(xmax,dt) > tol
        xmax = 1.2*xmax;
    end

    % Have a choice here for weight nodes:
    % (a) equally spaced, with vastly varying weights
    % (b) equal weights, with vastly varying space between each
    % (c) some compramise between (a) & (b)

    % Implement (a), the most straightforward method

    % nodes
    x = linspace(xmin,xmax,wN+2);
    x = x(2:wN+1);

    % Integral approximation
```

```
    w = PDF(x,dt);

    w = w / sum(w);
    % plot(x,w,'o');

    S = exp(x);

end % function
```

B.4.4 Least Squares Weighted by the Jump Amplitude PDF

```
% Hedge to minimize the quadratic jump risk error
% weighted by the jump amplitude distribution

% N is the number of hedging instruments (excluding underlying asset)
% Target option exprires at T
% Hedging options expire at u
% Time now is t
% Time period over which to hedge is dt
% Asset price now is St

function [Kh Wh WS WB] = leastSquaresJumpRisk(N,t,dt,St,Ht,Sdelta,delta)
% Note: Sdelta and delta are not used by this function

    disp(sprintf('Jump Risk Strategy'))

    [Kh Wh WS WB] = leastSquares(N,t,dt,St,Ht,@jumpPDFapprox);

end % function


function [S w] = jumpPDFapprox(N,dt)
% Note: dt is not used by this function

    global mu del

    % An almost irrational number to ensure no node is at S=1
    Smax = 1.325467;
    Smin = Smax/(2*N-1);
    S = linspace(Smin,Smax,N);
    dS = ( S(N)-S(1) )/(N-1);

    % log-normal distribution
    w = exp(-(log(S)-mu).^2/(2*del^2))./(sqrt(2*pi)*del*S);

    % Normalize
    w = w/sum(w);

end % function
```

```
function [] = plotJumpPDF()

    global mu del

    N = 2^8;

    % Note: S(1) is approximately dS/2
    Smax = 1.6;
    Smin = Smax/(2*N);
    S = linspace(Smin,Smax,N);
    dS = ( S(N)-S(1) )/(N-1);

    % These two lines are equivalent:
    w = Normal( ( log(S)-(mu-del^2) ) / del );
    g = exp(-(log(S)-mu).^2/(2*del^2))./(sqrt(2*pi)*del*S);

    % Monte Carlo simulated jumps
    SMC = exp(mu + del*randn(5*1e5,1));
    [Smid density] = distribution(SMC,Smin,Smax,N);

    length(density)
    length(w)

    sum(density)
    sum(w)

    % Expectation
    sum(density.*Smid')
    sum(S.*w)
    expected = exp(mu+0.5*del^2)

    %plot(S,w,'.');
    hold on;
    plot(Smid, density,'.','MarkerSize',8);
    plot(S,g,'k-');
    plot([expected expected], [0 10], 'k--');
    xlabel('Y','FontSize',16);
    ylabel('PDF','FontSize',16);
    axis([0 S(N) 0 (max(g)+0.01)]);

end % function
```

B.4.5 Gauss-Hermite Hedging Strikes and Weights

```
% Generate hedge strikes and weights according to Carr & Wu
% using Gauss-Hermite quadrature

% Target option exprires at T
% Hedging options expire at u
```

```
% Time now is t
% Time period over which to hedge is dt
% Value of the hedge at time t is Ht

function [Kh Wh WS WB] = GaussHermiteHedge(N,t,dt,St,Ht,Sdelta,delta)
% create strikes and weights for the hedging options
% according to Carr & Wu (2004)

% Note: Sdelta and delta are not used by this function

    global r sigma lambda mu del T K ttype htype u M tN jMax

    disp(sprintf('Gauss Hermite Strategy'));

    after = T-u;

    % raw nodes and weights
    [x w] = quadrature(N);

    % variance (including jumps)
    var = sigma^2 + lambda*(mu^2 + del^2);

    % moneyness of the target option
    money = K./St;

    % for each hedging option
    for n = 1:1:N

        % moneyness of a hedging option
        moneyH = money*exp( x(n)*sqrt(2*var*after) - (r+var/2)*after );

        % Gamma of the target option at time t+dt
        gamma = eMerton(lambda,mu,del,r,sigma,T-t-dt,...
                        moneyH,money,'gamma',ttype,jMax);

        % hedging weights
        Wh(:,n) = gamma .* moneyH *sqrt(2*after*var) * w(n) / exp(-x(n)^2);
        % According to Carr and Wu, p.42, for the Black-Scholes world:
        % Wh = 1/sqrt(pi)*w;

        % strike = asset price * hedge moneyness
        Kh(:,n) = St.*moneyH;

    end

    % for each simulation
    for i = 1:1:length(St)

        % cost of hedging options
        cost = sum( Wh(i,:).* eMerton(lambda,mu,del,r,sigma,u-t,St(i),...
                    Kh(i,:),'value',htype,jMax) );

        % Weight in the underlying asset and bonds
        WS(i,1) = 0;
```

```
        % WB represents the Gauss-Hermite error, so should be small if lambda == 0
        WB(i,1) = Ht(i) - cost;

    end

    size(Kh, Wh, WS, WB)

end % function
```

B.4.6 Gauss-Hermite Quadrature

```
function [x w] = quadrature(N)
% Gauss-Hermite quadrature nodes and weights

% For use in approximating int_{-infty}^{infty} f(x) e^{-x^2} dx
% using the Guass-Hermite quadrature rule

    x = roots(hermite(N));
    x = sort(x);
    w = hermiteWeight(N,hermite(N-1),x);

end % function


function [polynomial] = hermite(N)
% Hermite polynomial of degree N (an integer)

    % Hermite polynomials ( order, coefficient of x^j )
    H = zeros(N+1,N+1);

    H(1,1) = 1;
    if N >= 1
        H(2,2) = 2;
    end % if

    if N >= 2
        for n = 3:1:N+1

            H(n,1) = - 2*(n-2)*H(n-2,1);
            for j = 2:1:N+1 % coefficient of x, x^2, ..., x^N
                % H_n = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), with n in [0,N]
                H(n,j) = 2*H(n-1,j-1) - 2*(n-2)*H(n-2,j);
            end % for

        end % for
    end % if

    polynomial = H(N+1,:);

    % Reverse the vector so it is in the required polynomial format
```

```
    polynomial = rot90(speye(N+1)) * polynomial';

end % function


function [weight] = hermiteWeight(N, Hprev, x)
% Hnext means the Hermite polynomial of order N+1

    if 1 < N
        weight = 2^(N-1)*factorial(N)*sqrt(pi) / N^2 ./ (polyval(Hprev,x)).^2;
    else % N == 1
        weight = 2^(N-1)*factorial(N)*sqrt(pi) / N^2 ./ 1;
    end % if

end % function


function [] = testHermite()
% Testing Hermite Polynomials
% This appears to be consistent with:
% http://mathworld.wolfram.com/Hermite-GaussQuadrature.html

    N = 3;
    H = hermite(N);
    x = roots(H);

    w = hermiteWeight(N,hermite(N-1),x);
    plot(x,w,'x');
    axis([-inf inf 0 inf]);

end % function
```

B.5 Hedging Experiments

B.5.1 The Main Calling Function for the Experiments

```
% The overall client of hedging functions

% Hedging of European vanilla options
% under Merton's jump-diffusion

% Hedging instruments expire at time u
% Target option expires at time T
% 0 <= t <= u < T

% Assume dividend yield q = 0 always

function [] = client()
```

```
    format compact;
    clear all;
    close all;

    global S0 T u N M KH WH WS WB Smin Smax NS ttype thedger Vhedger Shedger

    initializeParameters();

    NS = 2^7;

    strategy = { @deltaHedge, @GaussHermiteHedge,...
                 @leastSquaresError, @leastSquaresJumpRisk};

%    strategySurface(NS, strategy{2}, 'GH.mat');
%    load('GH.mat');

%    strategySurface(NS, strategy{3}, 'LSE.mat');
    load('LSE.mat');
%    whos

%    strategySurface(NS, strategy{4}, 'LSJ.mat');
%    load('LSJ.mat');

%    hedgerSurface('hedgers')
    load('hedgers');

    % LSJ.mat APPEARS INCORRECT!

    [mean stdev skew kurt lowerCentPC lowerDecaPC lowerPC lower5PC...
      upper5PC upperPC upperDecaPC upperCentPC] = compareDynamic();
% = compareVaryingHedgers(1);

    outputResults(mean, stdev, skew, kurt, lowerCentPC, lowerDecaPC,...
       lowerPC, lower5PC, upper5PC, upperPC, upperDecaPC, upperCentPC);


%     comparePayoffs(strategy{3},t,dt,St);


      % Relative P&L over one time horizon

%     for j = 1:1:3
%         windowPanes(3, 2, 2*j-1, 0.04, 0.08)
%         compareDifference(strategy{j+1}, t, dt, St, [0 1 3 5], (128/u)^0.1*0.04);
%
%         windowPanes(3, 2, 2*j, 0.04, 0.08)
%         compareDifference(strategy{j+1}, t, dt, St, [5 15], sqrt(128/u)*0.0004);
%     end

end % function


function [] = outputResults(mean, stdev, skew, kurt, lowerCentPC, lowerDecaPC,...
                  lowerPC, lower5PC, upper5PC, upperPC, upperDecaPC, upperCentPC)
```

```
% output a table of results

    fprintf(1,'%s\t', 'Mean');
    fprintf(1,'%1.6f\t',mean);
    fprintf(1,'\n');

    fprintf(1,'%s\t', 'Standard Deviation');
    fprintf(1,'%1.6f\t',stdev);
    fprintf(1,'\n');

    fprintf(1,'%s\t', 'Skew');
    fprintf(1,'%1.6f\t',skew);
    fprintf(1,'\n');

    fprintf(1,'%s\t', 'Kurtosis');
    fprintf(1,'%1.6f\t',kurt);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '0.01%');
    fprintf(1,'%1.6f\t',lowerCentPC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '0.10%');
    fprintf(1,'%1.6f\t',lowerDecaPC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '1.00%');
    fprintf(1,'%1.6f\t',lowerPC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '5.00%');
    fprintf(1,'%1.6f\t',lower5PC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '95.00%');
    fprintf(1,'%1.6f\t',upper5PC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '99.00%');
    fprintf(1,'%1.6f\t',upperPC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '99.90%');
    fprintf(1,'%1.6f\t',upperDecaPC);
    fprintf(1,'\n');

    fprintf(1,'%s\t', '99.99%');
    fprintf(1,'%1.6f\t',upperCentPC);
    fprintf(1,'\n');

end % function
```

B.5.2 Parameter Values

```
% The entire parameter set used by lots of functions
% Defining the price process, the option and other global constants

% These are not intended to be changed by other functions
% and would preferably be constants

function [] = initializeParameters()

    global r sigma S0 lambda mu del T K ttype htype u N M tN jMax Kh...
           rebPower rebNt rebN rebdt Nt

    % The parameters are as in p.26, Calibration & Hedging...

    % Underlying parameters
    r     = 0.05;
    sigma = 0.2;
    S0    = 1;
    St    = S0;

    % Jump parameters
    lambda = 0.1
    mu     = -0.92;
    del    = 0.425;

    % Target Option parameters
    T = 0.5;
    K = 1;
    ttype = 'put';

    % Hedging option parameters
    N = [0 1 2 3 5 10 20]; % number of options
    u = 0.25; % expiry
    htype = 'put';
    Kh = [1 0.9 1.1 0.8 1.2 0.7 1.3 0.6 1.4 0.5 1.5 0.95...
            1.05 0.85 1.15 0.75 1.25 0.65 1.35 0.55 1.45];

    % Number of simulations
    M =  1e5

    % Number of rebalance intervals
    rebPower = 0:5; % 0:2;
    rebNt = 2.^rebPower;
    rebN  = length(rebPower);
    rebdt = u./rebN;

    % Number of intervals for the sample paths
    Nt = 2^7;

end % function
```

B.5.3 Comparing the Value of Hedging and Target Options

```
% plot the payoffs of strategies for varying N

function [] = comparePayoffs(strategy, t, dt, St)

    global r sigma S0 lambda mu del T K ttype htype u N M tN

    disp(sprintf('PAYOFFS'));

    % Target option value = initial hedging funds
    Ht = eMerton(lambda,mu,del,r,sigma,T-t,S0,K,'value',ttype);

    % Precompute delta at each rebalance time
    [Sdelta delta] = eMertonSurface(lambda,mu,del,r,sigma,...
                     T,S0,K,'delta',ttype,[t u]);
    % Make delta have tN intervals
    delta(:,tN+1) = zeros(length(Sdelta),1);

    for n = 1:1:length(N)
        disp(sprintf('STATIC with %i hedging instrument(s)', N(n)));

        [Kh Wh WS WB] = strategy(N(n),t,dt,St,Ht,Sdelta,delta);

        subplot(2,2,n);
        plotHedgePayoff(Kh, Wh, WS, WB, t, dt);
        title( strcat('Static Hedge with...
             [', num2str(N(n)), '] hedging instrument(s)'), 'FontSize', 14 );

    end % for

end % function


function [] = plotHedgePayoff(Kh, Wh, WS, WB, t, dt)
% Compare hedge payoff functions to value of target at t = u

% Kh and Wh are vectors, WS is a scalar
% WS represents the weight in the asset S

    global r sigma S0 lambda mu del T K ttype htype u M tN jMax

    if nargin < 3
        WS = 0;
    end

    N = length(Kh);

    Send = 0.5:0.05:1.5;
    Hend = evaluate(Send,t,dt,Kh,Wh,WS,WB);

    % Target option value at t = T-t-dt
    Vend = eMerton(lambda,mu,del,r,sigma,T-t-dt,Send,K,'value',ttype);
```

```
    plot(Send,Hend, 'k-');
    hold on;
    plot(Send,Vend,   'k--', 'LineWidth', 2);
    for n =1:1:length(Kh)
        plot([Kh(n) Kh(n)], [0 max(Vend)], ':', 'LineWidth', 2);
    end
    hold off;
    legend('hedge', 'target');
    xlabel('S', 'FontSize', 14);
    ylabel('Value', 'FontSize', 14);
    axis([Send(1) Send(length(Send)) 0 inf])

end % function
```

B.5.4 $\Delta_J\Pi$ as a Function of S for Several Strategies

```
% compare plots of the value of the hedge - target against S
% for varying N at time t+dt

function [] = compareDifference(strategy, t, dt, St, N, ymax)

    global r sigma S0 lambda mu del T K ttype htype u M tN jMax

    disp(sprintf('DIFFERENCES'));

    % Target option value = initial hedging funds
    Ht = eMerton(lambda,mu,del,r,sigma,T-t,S0,K,'value',ttype);

    % Precompute delta at each rebalance time
    [Sdelta delta] = eMertonSurface(lambda,mu,del,r,sigma,T,...
                                    S0,K,'delta',ttype,[t u]);
    % Make delta have tN intervals
    delta(:,tN+1) = zeros(length(Sdelta),1);

    Smin = 0.6;
    Smax = 1.4;
    Send = linspace(Smin, Smax, 129);

    % Target option value at time T-t-dt
    Vend = eMerton(lambda,mu,del,r,sigma,T-t-dt,Send,K,'value',ttype);

    for n = 1:1:length(N)
        if( N(n) == 0 )
            disp(sprintf('Delta hedging'));
            strategyNow = @deltaHedge;
        else
            disp(sprintf('Hedge with %i option(s)', N(n)));
            strategyNow = strategy;
        end % if
```

```
        [Kh Wh WS WB] = strategyNow(N(n),t,dt,St,Ht,Sdelta,delta);

        % Check delta neutrality at time t
        % Hdelta = evaluateDelta(S0,t,0,Kh,Wh,WS,WB)
        % Vdelta = eMerton(lambda,mu,del,r,sigma,T-t,S0,K,'delta',ttype,jMax)

        % Value of the hedge at time T-t-dt
        Hend = evaluate(Send,t,dt,Kh,Wh,WS,WB);

        % Value of the portfolio at time T-t-dt
        Pi(:,n) = Hend-Vend;

        labels{n} = strcat(num2str(N(n)), ' hedging option(s)');

    end % for

    % Plot differences

    set(0,'DefaultAxesLineStyleOrder','-|--|-.|:',...
          'DefaultAxesColorOrder',[0 0 0]);
    plot(Send, Pi,'LineWidth',2);
    hold on;
    plot([Smin Smax], [0 0], 'k-');
    hold off;
    legend( labels );
    xlabel('S','FontSize',16);
    ylabel('\Delta \Pi','FontSize',16);
    %title( strcat('Time Period = ', num2str(dt)), 'FontSize', 14 );
    axis([Smin Smax -ymax ymax])

    y_formatstring = '%5.2f';
    ytick = get(gca,'ytick');
    for i = 1:length(ytick)
        yticklabel{i} = sprintf(y_formatstring,ytick(i));
    end
    set(gca,'yticklabel', yticklabel)

end % function
```

B.5.5 Comparing the Relative P&L of Hedging Strategies

```
% plot the relative P&L for hedging strategies
% with varying rebalance intervals

function [mean stdev skew kurt lowerCentPC lowerDecaPC lowerPC lower5PC upper5PC...
          upperPC upperDecaPC upperCentPC] = compareDynamic()
% Compare relative P&L for different rebalance intervals

    global r sigma S0 lambda mu del T K ttype htype u M jMax rebPower...
           rebNt rebN rebdt Nt
```

```
% Maximum number of rebalance times
rebNtnow = rebNt(rebN-3);

% For plotting P&L distributions
PLmin =  -0.18;
PLmax =   0.18;
bins = 2^7;

% Index of the number of hedging instruments N
n = 6;

% Create simulated paths
% Use the same paths for all hedging strategies
% It is important that the number of time intervals is times(Ntimes)
[t X] = jumpDiffusionSpot(lambda,mu,del,r,sigma,u,S0,M,Nt);

% Choose nearest indices of t, X to rebalance times
indices = round( Nt/rebNtnow*(1:(rebNtnow+1)) );
indices = indices - indices(1) + 1;

t = t(indices);
X = X(:,indices);

% Precompute delta at each time interval (for delta hedging only)
% used for purely delta hedging
% [Sdelta delta] = eMertonSurface(lambda,mu,del,r,sigma,T,...
                                  S0,K,'delta',ttype,t);
% Make delta have rebNtnow+1 columns
% delta(:,rebNtnow+1) = zeros(length(Sdelta),1);
Sdelta = 0; delta = 0;

Su = exp(X(:,rebNtnow+1));

% Starting with the most rebalance intervals
% and halving during each loop
for i = rebN-3:-1:1

    rebNtnow = rebNt(i)
    rebalance(i) = rebNtnow;

    if i ~= rebN-3
        % Reduce length of t, X and delta to relevant size

        rebalanceIndex = 1:2:rebNt(i+1)+1;

        % Keep every element of t, X, Sdelta delta with an odd index
        t = t(  rebalanceIndex);
        X = X(:,rebalanceIndex);
        % delta  = delta(:,rebalanceIndex);
    end

    Hu = dynamic(t,X,n,Sdelta,delta);
```

```
        PL = computeRelPL(Su,Hu);

        [mean(i) stdev(i) skew(i) kurt(i) lowerCentPC(i)...
           lowerDecaPC(i) lowerPC(i) lower5PC(i) upper5PC(i)...
           upperPC(i) upperDecaPC(i) upperCentPC(i)] = stats(PL);
        [PLmid density(:,i)] = distribution(PL,PLmin,PLmax,bins);

        % Note difference in notation: N means time intervals here
        labelsN{i} = strcat( 'N = ', num2str(rebNtnow) );
    end % for

    % Plot P&L distribution

    figure
    set(0,'DefaultAxesLineStyleOrder','-|--|:|-.',...
          'DefaultAxesColorOrder', [0 0 0]);
    plot(PLmid,density,'LineWidth',2);
    axis([PLmin+0.03 PLmax-0.03  0 inf]);
    xlabel('Relative P&L','FontSize',14);
    ylabel('PDF','FontSize',14);
    legend( labelsN );
    title(strcat('Monte Carlo Results using [', num2str(M),...
                 '] Simulations'),'FontSize',14);

end % function
```

B.6 Hedging Simulation & Evaluation

B.6.1 Dynamic Hedging Simulation

```
% Dynamic hedging

function [V] = dynamic(t,X,n,Sdelta,delta)

    global r sigma S0 lambda mu del T K ttype htype u M tN KH WH WS WB...
           rebPower rebNt rebN rebdt

    % number of rebalance times
    rebNtnow = length(t)-1;

    % index of the number of rebalance times
    reb = rebPower(rebNtnow == rebNt)+1-rebPower(1);

    % assume t is a uniform grid
    dt = (t(rebNtnow+1)-t(1))/rebNtnow;

    S1 = exp( X(:,1) );
    V0 = eMerton(lambda,mu,del,r,sigma,T-t(1),S1,K,'value',ttype);
```

```
    for i = 1:1:rebNtnow

        % This line is fairly expensive and could be avoided
        % by using x = log(S) throughout
        St = exp( X(:,i) );

        if i == 1
           % Initial funds are equal to the value of the target
           Ht(:,i) = V0;
        else % 1 < i
           Ht(:,i) = evaluateApprox(St,t(i)-dt,dt,KHnow,WHnow,WSnow,WBnow);
        end

        % Kh and Wh are matrices with dimensions:
        % (simulation instance, hedging instrument)
        [KHnow WHnow WSnow WBnow] = getStrategy(reb,n,i,St,t(i),dt,Ht(:,i),V0);

     end

    % The last time interval
    Send = exp( X(:,rebNtnow+1) );
    V = evaluate(Send,t(rebNtnow+1)-dt,dt,KHnow,WHnow,WSnow,WBnow);

    Ht(:,rebNtnow+1) = V;

end % function
```

B.6.2 The Value of a Hedge

```
% the combined value of the hedge at time t+dt

function [V] = evaluate(Send,t,dt,Kh,Wh,WS,WB)
% Send may be a vector corresponding to each sample path
% Kh may be a matrix of strikes, where each row corresponds to each sample path

    global r sigma S0 lambda mu del T K htype u tN M

    % Assume length(Wh) == length(Kh)

    N = size(Wh)*[0; 1];

    V = zeros(size(Send));
    for n = 1:1:N

        % evaluate hedging options at time t+dt
        option = eMerton(lambda,mu,del,r,sigma,u-t-dt,Send,Kh(:,n),'value',htype);

        V = V + Wh(:,n) .* option;
    end % for
```

```
    V = V + Send.*WS + exp(r*dt)*WB;

end % function
```

B.6.3 The Approximate Value of a Hedge

```
% Approximate value of the hedge at time t+dt

function [V] = evaluateApprox(Send,t,dt,Kh,Wh,WS,WB)
% Send may be a vector corresponding to each sample path
% Kh may be a matrix of strikes, where each row corresponds to each sample path

    global r sigma S0 lambda mu del T K htype u tN M

    % Assume length(Wh) == length(Kh)

    N = size(Wh)*[0; 1];

    V = zeros(size(Send));
    for n = 1:1:N

        % evaluate hedging options at time t+dt
        option = getHedgerV(Send,t+dt,n);

        V = V + Wh(:,n) .* option;
    end % for

    V = V + Send.*WS + exp(r*dt)*WB;

end % function
```

B.6.4 The Combined Delta of a Hedge

```
% the combined delta of the hedge at time t+dt

function [V] = evaluateDelta(Send,t,dt,Kh,Wh,WS,WB)

    global r sigma S0 lambda mu del T K htype u tN M

    % Assume length(Wh) == length(Kh)

    N = size(Wh)*[0; 1];

    V = zeros(size(Send));
    for n = 1:1:N
```

```
        option = eMerton(lambda,mu,del,r,sigma,u-t-dt,Send,Kh(:,n),'delta',htype);

        V = V + Wh(:,n) .* option;
    end % for

    V = V + 1*WS + 0*WB;

end % function
```

B.6.5 The Relative Profit & Loss of a Portfolio

```
% The relative P&L for a hedge given the terminal asset price

function [relPL] = computeRelPL(Su,hedge)
% This should be called at time t = u
% Relative P&L = exp{-rT} Pi(T) / V(S_0, 0)

    global r sigma S0 lambda mu del T K ttype htype u M tN

    multiplier = exp(-r*u) / eMerton(lambda,mu,del,r,sigma,T,S0,K,'value',ttype);
    target = eMerton(lambda,mu,del,r,sigma,T-u,Su,K,'value',ttype);

    relPL = multiplier*(hedge-target);

end % function
```

B.7 Pre-computing Prices, Delta & Strategies

B.7.1 Storing Target Option Prices under Jump-Diffusion

```
% Create a surface of eMerton() evaluations - pre-compute points V(S,t)

function [S V] = eMertonSurface(lambda,mu,del,r,sigma,T,S0,K,opt,type,t)

% Note that the option will expire in time T-t
% and S is the current asset price

    Nt = length(t)-1;
    t = t(1:Nt);

    Smin = 0.05; Smax = 2;
    NS = 512; % 1024;
    S = linspace(Smin, Smax, NS+1);
```

```
%    tic
    for i = 1:1:Nt
        V(:,i) = eMerton(lambda,mu,del,r,sigma,T-t(i),S,K,opt,type);
    end % for
%    mertonRunTime = toc/(Nt*(NS1+1))

end % function


function [] = test_surface()

    % Underlying parameters
    r     = 0.05;
    sigma = 0.2;
    S0    = 1;

    % Jump parameters
    lambda = 0.1;
    mu     = -0.92;
    del    = 0.425;

    % Target Option parameters
    T = 1; % 0.5;
    K = 1;
    type = 'put';

    opt = 'delta';

    tN = 2^8;
    t = linspace(0,T,tN+1);

    [S V] = eMertonSurface(lambda,mu,del,r,sigma,T,S0,K,opt,type,t);
    NS = length(S)-1;
    Smin = S(1);
    Smax = S(NS+1);

    tests = 200;

    SI = 1 + randn(1,tests);
    SI(SI <= 0) = 1e-5;

 %   surf(t,S,delta)

    j = ceil(tN/2);

    plot(S,V(:,j),'k-');
    hold on;
    Saccurate = 0.05:0.05:2;
    Vaccurate = eMerton(lambda,mu,del,r,sigma,T-t(j),Saccurate,K,opt,type);
    plot(Saccurate,Vaccurate,'r-');

    Vtest = V(:,j);
```

```
    % Test evaluation of the polynomial runtime
    tic
    for i = 1:1:tests

        % manually programming nearest neighbour is quickest
        index = 1+ round( (SI(i)-Smin)/(Smax-Smin) * NS );

        if index > NS+1
            index = NS+1;
        elseif index < 1
            index = 1;
        end

        yi = Vtest(index);

    end % for
    NearestRunTime = toc/tests

    SI = SI(tests);
    Vnearest = Vtest(index)
    VExact   = eMerton(lambda,mu,del,r,sigma,T-t(j),SI,K,opt,type,jMax)

end % function
```

B.7.2 Storing Hedging Option Prices under Jump-Diffusion

```
% Create a hedger surface

function [] = hedgerSurface(filename)

    global r sigma S0 lambda mu del T K ttype N u htype Kh tN...
           rebPower rebNt rebN rebdt

    NN = length(N);

    Smin = 0.01; Smax = 3;

    % Precompute hedging prices
    NShedger = 2^11;
    Shedger = linspace(Smin,Smax,NShedger+1);
    Vhedger = zeros(N(NN),rebNt(rebN)+1,NShedger+1);
    thedger = linspace(0,u,rebNt(rebN)+1);

    for k = 1:1:N(NN)
        for i = 1:1:rebNt(rebN)+1
            Vhedger(k,i,:) = eMerton(lambda,mu,del,r,sigma,u-thedger(i),...
                                     Shedger,Kh(k),'value',htype);
        end
    end
```

```
    disp(sprintf('Writing'));

    save(filename, 'r','sigma','S0','lambda','mu',...
        'del','T','u','rebPower','rebNt','rebN','rebdt',...
        'NShedger','Smin','Smax','K','Kh',...
        'Shedger', 'Vhedger','thedger',...
        '-v6');

end % function
```

B.7.3 Requesting the Value of a Hedging Option

```
% Get eMerton() value of a hedging option (from precomputed Vhedger)

function [V] = getHedgerV(St,t,k)
% St is a vector current sample path asset prices
% i is the index of the time where t = linspace(0,u,rebNt(rebN)+1);
% k is the index of Kh, the strike of the hedger

    global Shedger Vhedger thedger

    % Use the nearest time
    Nt = length(thedger)-1;
    tmin = thedger(1);
    tmax = thedger(Nt+1);
    dt = (tmax-tmin)/Nt;
    i = round( (t-tmin)/dt ) + 1;

    Smin = Shedger(1);
    NS = length(Shedger);
    Smax = Shedger(NS);
    dS = (Smax-Smin)/NS;
    lower = Smin/dS;

    % Use nearest neighbour (for speed)
    j = round(St./dS - lower) + 1;
    j(j > NS+1) = NS+1;
    j(j < 1) = 1;

    V1 = Vhedger(k,i,j);

    V = zeros(length(St),1);
    V(:,1) = V1(1,1,:);

end % function
```

B.7.4 Storing Hedging Strategies under All Circumstances

```
% Create a surface of strategy() evaluations - pre-compute points

function [] = strategySurface(NS, strategy, filename)
% tN = a vector containg the number of rebalance times
% NS = asset price intervals
% strategy = function handle for the hedging strategy
% integralWeighting = function handle for the weighting

% This stores strikes, weights and asset weight as matrices
% with indexing: (rebN, N, t, S, option n)
% where rebN is the number of rebalance times
% and N is the number of hedging options

    global r sigma S0 lambda mu del T K ttype N u htype Kh tN...
           rebPower rebNt rebN rebdt

    % target option price at time 0
    V0 = eMerton(lambda,mu,del,r,sigma,T,S0,K,'value',ttype);

    NN = length(N);

    Smin = 0.01; Smax = 3;

    S = linspace(Smin, Smax, NS+1);

    KH = zeros(rebN, NN,rebNt(rebN),NS+1,N(NN));
    WH = zeros(rebN, NN,rebNt(rebN),NS+1,N(NN));
    WS = zeros(rebN, NN,rebNt(rebN),NS+1);
    WB = zeros(rebN, NN,rebNt(rebN),NS+1);

    Sdelta = 0;
    delta  = 0;

    % Assume we have funds equal to V0 at time t(i)
    % WB has to be computed at each rebalance to fit actual funds
    Ht = V0*ones(size(S));

    tic

    % Loop through number of rebalance intervals
    for reb = 1:rebN

        reb

        t = linspace(0,u,rebNt(reb)+1);
        t = t(1:rebNt(reb));
        dt = u/(rebNt(reb));

        % Loop through number of hedging options
        for n = 1:NN
            n
```

```
            % Loop through rebalance times
            for i = 1:rebNt(reb)
                [KH(reb,n,i,:,1:N(n)), WH(reb,n,i,:,1:N(n)),...
                 WS(reb,n,i,:), WB(reb,n,i,:)]...
                    = strategy(N(n),t(i),dt,S,Ht,Sdelta,delta);
            end % for
        end % for
    end % for

    strategySurfaceRunTimeAverage = toc/(rebN*rebN/2*(NS+1))

    disp(sprintf('Writing'));

    % Save all data file
    % -v6 turns off data compression
    save(filename,'t','S','r','sigma','S0','lambda','mu',...
        'del','T','u','rebPower','rebNt','rebN','rebdt',...
        'NS','Smin','Smax','K','Kh',...
        'KH','WH','WS','WB',...
        '-v6');

end % function
```

B.7.5 Requesting a Hedging Strategy

```
% Get strategy solution (from precomputed KH, Wh, WS, WB)

function [KHout WHout WSout WBout] = getStrategy(reb,n,i,St,t,dt,Ht,V0)

% reb is the index of the number of rebalance intervals
% n is number of strikes
% i is time index (in accordance with the saved strategy surface)
% St is current asset price
% Ht are current funds
% Vt is price of target option originally at

    global  KH WH WS WB Smin Smax NS N

    dS = (Smax-Smin)/NS;

    Sindex = (St-Smin)./dS + 1;

    % Truncate to S domain
    Sindex(Sindex >= NS+1) = NS+1-1e-12;
    Sindex(Sindex <= 1)    = 1   +1e-12;

    before = floor(Sindex);
    after = before + 1;

    left  = Sindex - before;
```

```
    right = 1 - left;

    KHl1 = KH(reb,n,i,before,1:N(n));
    KHr1 = KH(reb,n,i, after,1:N(n));
    WHl1 = WH(reb,n,i,before,1:N(n));
    WHr1 = WH(reb,n,i, after,1:N(n));

    KHl = zeros(length(St),N(n));
    KHr = zeros(length(St),N(n));
    WHl = zeros(length(St),N(n));
    WHr = zeros(length(St),N(n));

    KHl(:,:) = KHl1(1,1,1,:,:);
    KHr(:,:) = KHr1(1,1,1,:,:);
    WHl(:,:) = WHl1(1,1,1,:,:);
    WHr(:,:) = WHr1(1,1,1,:,:);

    % Linear interpolation
    for i = 1:1:N(n)
        KHout(:,i) = right.*KHl(:,i) + left.*KHr(:,i);
        WHout(:,i) = right.*WHl(:,i) + left.*WHr(:,i);
    end

    WSl1 = WS(reb,n,i,before);
    WSr1 = WS(reb,n,i, after);
    WBl1 = WB(reb,n,i,before);
    WBr1 = WB(reb,n,i, after);

    WSl = zeros(length(St),1);
    WSr = zeros(length(St),1);
    WBl = zeros(length(St),1);
    WBr = zeros(length(St),1);

    WSl(:,1) = WSl1(1,1,1,:);
    WSr(:,1) = WSr1(1,1,1,:);
    WBl(:,1) = WBl1(1,1,1,:);
    WBr(:,1) = WBr1(1,1,1,:);

    % Linear interpolation
    WSout = right.*WSl + left.*WSr;
    WBout = right.*WBl + left.*WBr;

    % Cost of hedge without bonds at time t
    cost = evaluateApprox(St,t-dt,dt,KHout,WHout,WSout,0);

    % Compensate WB for profit/lost accured in hedging so far
    WBout = Ht - cost;

end % function
```

B.8 Fundamental Tools & Functions

B.8.1 Merton's Jump-Diffusion Transition PDF

```
% Transition PDF for Merton's jump diffusion

function [p] = PDF(x,dt)
% Probability Density Function
% Cont, Tankov (book): p.111
% x is log(S) at time t_{i+1} given x = 0 at time t_{i}
% dt is the time interval

    global r sigma S0 lambda mu del T K type u M tN

    N = ceil(60*lambda*T);

    % t is the time interval
    t = dt;
    sig2 = sigma^2;
    del2 = del^2;
    mu2 = mu^2;
    sig2t = sig2*t;
    tLambda = t*lambda;
    rm = r - 0.5*sig2 + 0.5*lambda*(mu2 + del2);
    rmt = rm*t;

    sum = 0;

    kFactorial = 1;
    tLambdaPowerK = 1;

    for k = 0:1:N
       if k > 0
           kFactorial = k*kFactorial;
           tLambdaPowerK = tLambda*tLambdaPowerK;
       end

       % Merton variance
       var = sig2t + k*(mu2+del2);

       Poisson = tLambdaPowerK/kFactorial;
       GaussianNum = exp( -(x-rmt-k*mu).^2 ./ (2*var) );
       GaussianDen = sqrt(2*pi*var);

       sum = sum + Poisson * GaussianNum ./ GaussianDen;
    end

    % normalization
    p = exp(-tLambda) * sum;

end % function
```

```
function [] = plotPDFs()
% Uses Monte Carlo simulation to create a density function
% and compares this against the analytical PDF()

    global r sigma S0 lambda mu del T K type u M tN jMax

    dt = u;
    tol = 1e-3;

    xmin = -1e-3;
    while PDF(xmin,dt) > tol
        xmin = 1.2*xmin;
    end

    xmax = 1e-3;
    while PDF(xmax,dt) > tol
        xmax = 1.2*xmax;
    end

    x = linspace(xmin,xmax,256);

    lambda0 = lambda;

    ymin = -1.5;
    ymax = 0.6;

    border = 0.01;
    spacing = 0.06;

    for i = 1:1:2

        if(i == 1)
            lambda = 0.2;
        else
            lambda = 0.4;
        end

        windowPanes(2, 2, 2*i, border, spacing);

        % Plot exact PDF
        p = PDF(x,dt);
        plot(p,x, 'k-');

        % Normalize
        p = p/sum(p);
        PDFprice = sum(payoff(exp(x),K,type).*p)
        merton = eMerton(lambda,mu,del,r,sigma,u,S0,K,'value',type)

        % Create paths
        [t X] = jumpDiffusionSpot(lambda,mu,del,r,sigma,dt,S0,M,tN);
        terminal = X(:,tN+1);
        [xmid density] = distribution(terminal, xmin, xmax, 128);
        density = density / sum(density) / (xmid(2)-xmid(1));
```

```
        % Plot Monte Carlo PDFs
        hold on;
        plot(density,xmid,'k--','LineWidth', 2);
        xlabel('PDF','FontSize', 16);
        % ylabel('x','FontSize', 16);
        legend('Analytical', strcat(num2str(M),' Monte Carlo simulations'));
        axis([0 1.7 ymin ymax]);

        % Plot some sample paths
        windowPanes(2, 2, 2*i-1, border, spacing);
        plot(t,X(1:100,:),'k');
        xlabel('t','FontSize', 16);
        ylabel('x','FontSize', 16);
        axis([0 u ymin ymax]);

    end

end
```

B.8.2 The Payoff of Several Standard Options

```
% Payoff of some standard options

function [V] = payoff(S,K,type)
% type == 'call' or 'put' or 'straddle'

    switch type
        case 'call'
            V = max(S-K,0);
        case 'put'
            V = max(K-S,0);
        case 'straddle'
            V = abs(S);
    otherwise
        warning('Invalid type -- must be ''call'', ''put'' or ''straddle''')
        V = zeros( max( length(S), length(K) ) );
    end % switch

end % function
```

B.8.3 Approximating a Distribution from Sample Data

```
% Create an approximation to the PDF for data x

function [xmid density] = distribution(x,xmin,xmax,N)
% N is the number of bins

    bins = linspace(xmin, xmax, N+1);
    dx = (xmax-xmin)/N;

    denom = length(x)*dx;

    frequency = zeros(N,1);
    remaining = length(x);

    % Originally, the inside of the loop was:
    % frequency(i) = sum( bins(i) <= x & x < bins(i+1) );
    % This version is much quicker

    for i = 1:1:N
        l = x < bins(i);
        lCount = sum(l);

        r = bins(i+1) <= x;
        rCount = sum(r);

        frequency(i) = remaining - lCount - rCount;

        x = x(l|r);
        remaining = lCount+rCount;
    end

    xmid = linspace(bins(1)+dx/2, bins(N+1)-dx/2, N);

    density = frequency/denom;

    % The proportion of the data in the range [xmin, xmax]
    % sum(density*dx)

end % function
```

B.8.4 Statistical Measurements of a Dataset

```
% some standard statistics to summarize the data x

function [mean stdev skew kurt lowerCentPC lowerDecaPC...
lowerPC lower5PC upper5PC upperPC upperDecaPC upperCentPC] = stats(x)
% x is a vector containing the data

    N = length(x);
```

```
    mean  = sum(x) / N;
    stdev = sqrt(sum((x-mean).^2) / N);
    skew  = sum((x-mean).^3) / N;
    kurt  = sum((x-mean).^4) / N;

    x = sort(x);

    upperCentPC = x( min( ceil(N*9999/10000),N) );
    lowerCentPC = x( max(floor(N/10000),1) );

    upperDecaPC = x( min( ceil(N*999/1000),N) );
    lowerDecaPC = x( max(floor(N/1000),1) );

    upperPC = x( min( ceil(N*99/100),N) );
    lowerPC = x( max(floor(N/100),1) );

    upper5PC = x( min( ceil(19*N/20),N));
    lower5PC = x( max(floor(N/20),1) );

end % function
```

B.8.5 A Simple Sub-Plotting Tool

```
% Divide a figure into many plots

function [] = windowPanes(rows, cols, pane, border, spacing)

% Panes (plots) are in this order:
% 1 2 3
% 4 5 6

    width = (1 - 2*border - (cols+1)*spacing)/cols;
    height =(1 - 2*border - (rows+1)*spacing)/rows;

    row = 1+floor((pane-1)/cols);
    col = pane - (row-1)*cols;

    x = border + col*spacing + (col-1)*width;
    y = border + (rows-row+1) * spacing + (rows-row)*height;

    subplot('Position', [x y width height]);

end % function
```

Bibliography

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

[2] D. S. Bates. Pricing options under jump-diffusion processes. working paper 37-88. 1988. Rodney L. White Center for Financial Research, The Wharton School, University of Pennsylvania.

[3] A. Bjork. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996. ISBN 0-89871-360-9.

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

[5] M. Briani, R. Natalini, and G. Russo. Implicit-explicit numerical schemes for jump-diffusion processes. 1991.

[6] P. Carr and L. Wu. Static hedging of standard options. working paper. 2004. Courant Institute, New York University.

[7] R. Cont and P. Tankov. *Financial Modelling with Jump Processes*. Chapman & Hall/CRC, 2004. ISBN 1-58488-413-4.

[8] R. Cont, P. Tankov, and E. Voltchkova. Hedging with options in models with jumps. 2005. Working paper.

[9] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, 1988. ISBN 0-89871-360-9.

[10] Y. d'Halluin, P.A. Forsyth, and K.R. Vetzal. A penalty method for american options with jump diffusion processes. *Numerische Mathematik*, **97:2**:321–352, 2004.

[11] Y. d'Halluin, P.A. Forsyth, and K.R. Vetzal. Robust numerical methods for contingent claims under jump-diffusion processes. *IMA Journal of Numerical Analysis*, **25**:87–112, 2005.

[12] J. D. Duffy. *Finite Difference Methods in Financial Engineering, A Partial Differential Equation Approach*. Wiley, 2006. ISBN 0-470-85882-6.

[13] P.A. Forsyth and K.R. Vetzal. Quadratic convergence of a penalty method for valuing american options. *SIAM Journal of Scientific Computation*, **23**:2096–2123, 2002.

[14] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, 2004. ISBN 0-387-00451-3.

[15] A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, **58**:83–90, 1971.

[16] A. G. Hawkes and D. Oakes. A cluster process representation of a self-exciting process. *J. Appl. Prob.*, **11**:493–503, 1974.

[17] C. He, J.S. Kennedy, T. Coleman, P.A. Forsyth, Y. Li, and K. Vetzal. Calibration and hedging under jump diffusion. 2005.

[18] P. Hewlett. Clustering of order arrivals, price impact and trade path optimisation. 2006. Oxford University.

[19] S. Howison, P. Wilmott, and J. Dewynne. *The Mathematics of Financial Derivatives*. Cambridge University Press, 1995. ISBN 0-521-49789-2.

[20] J. C. Hull. Options, futures and other derivatives. 2006. Sixth Edition. ISBN 0-13-149908-4.

[21] M.S. Joshi. *The Concepts and Practice of Mathematical Finance*. Cambridge University Press, 2003. ISBN 0-521-82355-2.

[22] Y. M. Kabanov and M. M. Safarian. On Leland's strategy of option pricing with transactions costs. *Finance and Stochastics*, pages 239–250, 1997.

[23] J.S. Kennedy, P.A. Forsyth, and K.R. Vetzal. Dynamic hedging under jump diffusion with transaction costs. 2006. Working Paper.

[24] V. Kosmetatos. A numerical method for pricing american options with jump diffusion. 2003. MSc in Mathematical Modelling and Scientific Computing.

[25] H.E. Leland. Option pricing and replication with transactions costs. *The Journal of Finance*, **40**:1283–1301, 1985.

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

[27] W.E. Milne. *Numerical Solution of Differential Equations*. 1970. ISBN 0-486-62437-4.

[28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999. ISBN 0-387-98793-2.

[29] A. Prideaux. Finite difference methods for pricing american put options. 2005. MSc Mathematical Modelling and Scientific Computing, University of Oxford.