Valuing Hedge Fund Fees

by

Li Xiao

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2006
© Li Xiao 2006

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis applies a Partial Integral Differential Equation model, along with a Monte Carlo approach to quantitatively analyze the no arbitrage value of hedge fund performance fees. From a no-arbitrage point of view, the investor in a hedge fund is providing a free option to the manager of the hedge fund. The no-arbitrage value of this option can be locked in by the hedge fund manager using a simple hedging strategy. Interpolation methods, grid construction techniques and parallel computation techniques are discussed to improve the performance of the numerical methods for valuing this option.

Acknowledgments

I would like to thank my supervisors Dr. Peter Forsyth and Dr. Ken Vetzal for their patience and guidance with my work.

Thanks also go to my family for supporting me throughout the years, and to my friends and scicom lab mates.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hedge funds are pooled investments that attempt to obtain superior returns for their mostly wealthy investors. The fund manager runs the fund and collects fees to compensate for expenses, management fees and superior performance. Hedge funds have grown rapidly. Since the late 1980s, the number of hedge funds has risen by more than 25 percent per year. The rate of growth in hedge fund assets has been even more rapid. At the end of 2004, there were more than 8000 hedge funds managing a total of almost one trillion dollars [40]. Hedge funds have come to play a large role in financial markets, there has been increasing attention focused on their management and investment practices.

The ideal fund structure aligns investors' goals with fund managers' incentives. Generally, four basic mechanisms determine this alignment: market forces, government regulation, incentive contracts and ownership structure [1]. Mutual funds generally emphasize the first two factors. In contrast, hedge funds tend to rely more heavily on the latter two. Hedge fund incentive plans are primarily bonus plans. Hedge fund managers typically receive a fraction of the fund's return each year in excess of the high-water mark. The high-water mark for each investor is the maximum share value since his or her investment in the fund. We refer to this fraction of the fund's return as a *performance fee*. These performance fees generally range from 15 percent to 25 percent of the new profit earned each year. Obviously, this performance fee can be considered a call option on the profits associated with managing other people's money, since the fee structure gives the managers the positive fees with profits but no negative fees with losses. In addition, managers charge a regular annual fee from one percent to two percent of portfolio assets to cover overhead. Hedge fund managers are usually required to invest a significant amount

of their own wealth in the fund. The investors of hedge funds believe that the combination of incentive bonus plans and managerial investment can move managerial effort closer to the optimal level.

Hedge funds have been in existence for fifty years. However, the hedge fund industry has remained opaque to the general investing public. The main reason may be that hedge funds are not required to report their returns to the public, so obtaining data for empirical studies is difficult. At present, academic research into hedge funds mainly focuses on the following three directions.

The first direction documents the general characteristics of fund performance. These papers use empirical data to analyze hedge fund performance characteristics. Since integrated statistical history data on hedge fund performance is not available, there exist various biases in existing hedge fund databases. Fung and Hsieh [2] review these biases in these databases. Stephen et al. [29] try to determine, using existing hedge fund performance data, whether there are differential skills which enable the hedge fund managers to obtain persistent performance. In [3], the authors provide a rationale on how hedge funds are organized, and provide some insight as to why hedge fund performance is different from traditional mutual funds. Some previous papers have investigated the relationship between hedge fund performance and standard asset indices [4, 5]. Fung and Hsieh [14] use lookback straddles to model trend-following strategies, and show that they can explain trend-following funds' return better than standard asset indices.

Another research direction focuses on the effect of this bonus contract upon the optimal dynamic investment strategies of hedge fund managers. The research along this line can be divided into two classes. The first class uses mathematical models to analyze the behavior of hedge fund managers under a bonus plan. It is postulated that the managers always attempt to maximize their expected utility. This research includes works by Goetzmann et al. [7], Griblatt and Titman [8] and Carpenter [9]. All of this work shows analytically that the portfolio variance of hedge funds increases due to the option-like feature of the high-water mark contract. In particular, Carpenter [9] identifies a strategy using the variance of the portfolio that depends upon the distance of the net asset value of the portfolio from the high-water mark. Out of the money managers have a strong incentive to increase variance, while in the money managers prefer lower risk. Carpenter [10] and Basak et al. [11] focus primarily on the optimal dynamic investment policy for a risk averse fund manager compensated with a call option on the assets under manage-

ment. Another area of research attempts to observe the actual effect of the bonus plan on hedge fund managers' strategy from empirical data of hedge funds, such as in Stephen et al. [6]. They found little evidence that managers take actions such as increasing risk when their option-like position is out of money.

Another body of literature focuses on valuing the option-like performance fees on hedge fund assets. Goetzmann et al. [12] consider the hedge fund performance fee contract as a perpetual contract with a path-dependent payoff unless the fund is closed by poor returns. The authors attempt to value this contract from the investor's point of view. The payoff at any time depends on the high-water mark that is related to the maximum asset value achieved. Since the information about the portfolio is not available for the investors, it is impossible to replicate the payoff of the contract. Goetzmann et al. use a martingale framework to derive the value equation.

In this thesis, we will value the option-like performance fees of hedge funds. We consider this problem from the manager's point of view. Note that the manager has the fund portfolio information which is not available to the investors. Suppose the asset of a hedge fund follows geometric Brownian motion. Imagine that the hedge fund manager does nothing but follows a delta hedge strategy. He can always receive the no arbitrage value from the bonus contract. As a result, when an investor puts money into a hedge fund, he actually provides a free option to the manager of the hedge fund. The manager can receive the option value with certainty. In this thesis, we will quantify the value of this free option.

In [12], the authors suppose that the hedge fund asset follows geometric Brownian motion, the high-water mark changes continuously and the hedge fund has no contractual termination unless it is closed by poor performance. In this thesis, the asset is assumed to follow geometric Brownian motion with Poisson jumps [30], which is a more realistic model for hedge fund returns. We also assume that the high-water mark changes discretely, and that there is contractual termination. There is also some difference between the bonus contract in [12] and this thesis. In [12], the payoff of the bonus contract depends on the high-water mark. In this work, the payoff of the bonus contract depends on both a high-water mark and a hurdle rate. The hurdle rate is also called the required rate of return. In this work, the manager can receive the performance fees only if the asset is above the maximum value of high-water mark and the initial value of asset times the hurdle rate. In addition, both [12] and this thesis suppose that if the asset price falls to a constant fraction

of the high-water mark then the investor withdraws all his money and there are no further costs or fees.

The option value of the hedge fund fee can be formulated as a path-dependent option pricing problem [16, 24]. In the process of solving a path-dependent option pricing problem, we must frequently interpolate the numerical solution. The interpolation methods and grid construction techniques are very important in order to improve the performance of the numerical method. In this thesis, we analyze and provide some guidelines for these interpolation methods and grid construction techniques.

Generally, solving a path-dependent pricing problem requires a large amount of computation, and parallel computation can greatly improve the efficiency of computation. Fortunately, the present algorithm used in solving path-dependent pricing problems can be easily modified to run efficiently on multiprocessor architectures. In this thesis, we analyze parallel computation using the μC++ language and OpenMP.

In general, the main results of this thesis are

- We provide a mathematical model to quantitatively analyze the no arbitrage value of the option-like hedge fund performance fees.
- We analyze interpolation methods and grid construction techniques which can improve the performance of the numerical method.
- We test the performance of the path-dependent pricing method in a parallel processing environment.

The outline of this thesis is as follows. In Chapter 2, we describe a mathematical model for the option-like performance fee. Chapter 3 discusses the interpolation methods and grid construction techniques that are important to improve the efficiency of the computation. Chapter 4 shows a robust and reliable numerical method for the option-like performance fee problem. In Chapter 5, we value the performance fees using the Monte Carlo (MC) method and provide an accurate and fast MC algorithm to value a barrier option performance fee with jump diffusion. Chapter 6 shows a convergence rate tests and a comparison of pricing using MC vs. numerical PIDE methods. In Chapter 7, we use the mathematical model provided

in Chapter 2 to analyze the option-like performance fee. Chapter 8 presents methods of parallelizing the existing option pricing library using the μC++ language and OpenMP in a multiprocessor computer in order to obtain better performance. Finally, Chapter 9 draws some conclusions, and lists possible areas for future work.

Chapter 2

Mathematical Model

2.1 The Evolution Equations

Let S be the balance in an investor's account, and let $\hat{S}$ be the price of the actual underlying asset of the hedge fund. We assume the existence of a risk free money market account, for which the rate of return is $r(t)$ and that S follows the stochastic differential equation in the real world probability measure:

$$dS = (\mu - m_{total})Sdt + \sigma Sdz + (J-1)Sdq. \tag{2.1}$$

The actual underlying asset $\hat{S}$ satisfies the following random walk:

$$d\hat{S} = \mu\hat{S}dt + \sigma\hat{S}dz + (J-1)\hat{S}dq, \tag{2.2}$$

where

μ is the drift rate.

dq is an independent Poisson process, $= \begin{cases} 0 & \text{with probability } 1-\lambda^P dt. \\ 1 & \text{with probability } \lambda^P dt. \end{cases}$

$J-1$ is an impulse function producing a jump from S to JS.

σ is the volatility.

dz is the increment of a Wiener process.

m_{total} is the proportional rate at which management fees for the underlying fund are deducted from the investor's account.

The management fee is composed of two parts:

$$m_{total} = m_c + m_r, \tag{2.3}$$

where m_c is used to partly compensate the manager's work, m_r is used to maintain the daily operation of the hedge fund. Obviously if the dollar value of the balance in investor's account and underlying asset is the same at t_0 then at time t we have

$$\hat{S} = e^{m_{total}(t-t_0)}S. \tag{2.4}$$

Let $V(S,t)$ be the value of the cash flows to the manager of the hedge fund. These cash flows include m_c and performance fees, but exclude the expense m_r. Consider the viewpoint of a hedge fund manager who sets up a portfolio that consists of a long position in the cash flow security and a short position, x_1, in the actual underlying asset. The value of this portfolio, $\Pi(S,t)$, is given by,

$$\Pi(S,t) = V(S,t) - x_1\hat{S}. \tag{2.5}$$

The change in the value of this portfolio over the interval $t \to t+dt$ is given by,

$$d\Pi_{total} = d\Pi_{Brownian} + d\Pi_{jump}. \tag{2.6}$$

Using Ito's lemma we have

$$\begin{aligned} d\Pi_{Brownian} =& [V_t + (\mu - m_{total})SV_S + \frac{1}{2}\sigma^2 S^2 V_{SS}]dt \\ &+ \sigma S V_S dz - x_1(\mu\hat{S}dt + \sigma\hat{S}dz) + m_c S dt, \\ d\Pi_{jump} =& [V(JS,t) - V(S,t)]dq - x_1(J-1)\hat{S}dq, \end{aligned} \tag{2.7}$$

where $m_c S dt$ is the remain part of the continuous fee (m_{total}) exclude the overhead part (m_r). To eliminate the diffusion risk over this infinitesimal interval, the manager can choose to hold the position:

$$x_1 = \frac{S}{\hat{S}}V_S. \tag{2.8}$$

Substituting (2.8), (2.7) into (2.6), we have

$$d\Pi_{total} = (V_t - m_{total}SV_S + \frac{1}{2}\sigma^2 S^2 V_{SS})dt + [V(JS,t) - V(S,t)]dq - V_S(J-1)Sdq + m_c S dt. \tag{2.9}$$

Taking the expected value of this change under the risk neutral measure (Q measure), and we obtain

$$\begin{aligned} E^Q[d\Pi_{total}] = (V_t - m_{total}SV_S + \frac{1}{2}\sigma^2 S^2 V_{SS})dt + E^Q[V(JS,t) - V(S,t)]E^Q[dq] \\ - V_S S E^Q[J-1]E^Q[dq] + m_c S dt, \end{aligned} \tag{2.10}$$

where we assume that the probability of the jump and the distribution of the jump size are independent. As a shortcut in the derivation, we take the expectation under the risk neutral measure to obtain the final pricing equation. However the manager could hedge the jump risk using the method discussed in [15]. Defining $E^Q[J-1] = \kappa$ and λ as the Q measure mean arrival rate of the Poisson process, then we have

$$\begin{aligned} E^Q[d\Pi_{total}] = (V_t - m_{total}SV_S + \frac{1}{2}\sigma^2 S^2 V_{SS})dt + E^Q[V(JS,t)]\lambda dt \\ - V(S,t)\lambda dt - V_S S\kappa\lambda dt + m_c S dt. \end{aligned} \tag{2.11}$$

The expected return of the portfolio under Q measure should be

$$E^Q[d\Pi_{total}] = r\Pi dt. \tag{2.12}$$

Then, from (2.11) and (2.12) we have

$$V_\tau = \frac{1}{2}\sigma^2 S^2 V_{SS} - (r+\lambda)V + V_S S(r - m_{total} - \kappa\lambda) + m_c S + \lambda \int_0^{+\infty} g(J)V(JS, T-\tau)dJ, \tag{2.13}$$

where

T		is the expiry/maturity date.
τ	$=$	$T - t$.
t		is current time.
$g(J)$		is the Q measure probability density function of the jump amplitude J such that, $\forall J, g(J) \geq 0$ and $\int_0^\infty g(J)dJ = 1$.

A common assumption is that $g(J)$ is log normal,

$$g(J) = \frac{\exp(-\frac{(\log(J)-\mu)^2}{2\gamma^2})}{\sqrt{2\pi}\gamma J}. \tag{2.14}$$

Note that all parameters (λ, μ, γ) are risk neutral since we take the expectation under Q measure in equation (2.10).

2.2 Cash Flow, State Variables and Updating Rules

In addition to the proportional fee m_c in equation (2.3), the hedge fund manager is entitled to a performance fee. Typically, the hedge fund manager receives a bonus

(usually quarterly or yearly) which is reflected in the performance of the fund. We will precisely define the incentive fee in the following.

Let $t_i, i = 1, 2, \cdots, N$, be observation dates, typically quarterly or yearly. The value of the amount in investor's account at the instant before the i^{th} observation date is denoted by $S(t_i^-)$. The value of the high-water mark at t_i^- is given by

$$H(t_i^-) = \max_{k=1,2,\cdots,i-1}[S(t_k^+)], \tag{2.15}$$

where $S(t_k^+)$ is the value in the investor's account at the instant after t_k. Define the following rates:

$$\begin{aligned} &p - \text{performance rate,} \\ &h - \text{hurdle rate (least required rate of return).} \end{aligned}$$

We assume for simplicity that the interval between observation dates is a constant, i.e. $\Delta t = t_{i+1} - t_i = constant$. The incentive fee paid to the hedge fund manager at t_i is then

$$P(t_i) = p \times \max\{S(t_i^-) - \max[S(t_{i-1}^+)(1 + h\Delta t), H(t_i^-)],\ 0\}. \tag{2.16}$$

Note we assume that the investor's account is adjusted to reflect the removal of the fee,

$$S(t_i^+) = S(t_i^-) - P(t_i). \tag{2.17}$$

In order to determine the no-arbitrage value of the incentive fee to the manager, we need to include extra state variables in pricing equation (2.13) to take into account the path dependent effects implied by equation (2.16). Let

$$\begin{aligned} S_{old}(t) &= S(t_{i-1}^+), & t_{i-1}^+ \le t \le t_i^-, \\ H(t) &= \max_{k=1,2,\cdots,i-1}[S(t_k^+)] & t_{i-1}^+ \le t \le t_i^-. \end{aligned} \tag{2.18}$$

Then, the no-arbitrage value of the incentive fees is given by $V = V(S, S_{old}, H, t)$, where V satisfies equation (2.13) between valuation dates. In addition, we need to determine the no-arbitrage jump condition at each valuation date t_i. Readers can refer to [16] for details concerning the jump condition.

Consider the i^{th} observation date t_i, let

$$S_{old}^+ = S_{old}(t_i^+), S_{old}^- = S_{old}(t_i^-), H^+ = H(t_i^+), H^- = H(t_i^-), S^+ = S(t_i^+), S^- = S(t_i^-). \tag{2.19}$$

Then we have the following jump condition at time t_i

$$V(S^+, S^+_{old}, H^+, t_i^+) = V(S^-, S^-_{old}, H^-, t_i^-) - P_i, \tag{2.20}$$

or

$$V(S^-, S^-_{old}, H^-, t_i^-) = V(S^+, S^+_{old}, H^+, t_i^+) + P_i, \tag{2.21}$$

where $P_i = P(t_i)$ is the incentive payment in equation (2.16).

Let $S = S^-, H = H^-, S_{old} = S^-_{old}$, then

$$V(S, S_{old}, H, t_i^-) = V(S^+, S^+_{old}, H^+, t_i^+) + P_i, \tag{2.22}$$

where

$$\begin{aligned} P_i &= p \times \max\{S - \max[S_{old}(1 + h\Delta t), H], 0\}, \\ S^+ &= S - P_i, \\ S^+_{old} &= S^+, \\ H^+ &= \max(H, S^+). \end{aligned} \tag{2.23}$$

We also assume that

$$V(S, S_{old}, H, T) = 0, \tag{2.24}$$

where T is the fund closing time (i.e. no further fees paid after $t = T$).

For computational purposes, we define a computational domain,

$$\begin{aligned} &0 \le S \le S_{max}, \\ &0 \le H \le H_{max}, \\ &0 \le S_{old} \le (S_{old})_{max}. \end{aligned} \tag{2.25}$$

Since the PIDEs are independent one dimensional problems, which communicate only by jump conditions at observation dates, we can discretize each one dimensional PIDE (in the S direction) independently.

2.3 Boundary Conditions

We also suppose that all the investors withdraw their money when the hedge fund asset price drops below a fraction of the high-water mark. Once this situation

occurs, there are no fees paid. For simplicity, we also suppose there is no cost when the investors withdraw their money. As a result, we have the following Dirichlet boundary condition [24],

$$V(S, S_{old}, H, t) = 0, \quad S \leq b \times H, \tag{2.26}$$

where b is a constant scalar. Effectively, $S = b \times H$ is a knock-out barrier, triggered when the liquidation level is met.

As $S \to 0$, equation (2.13) reduces to

$$V_\tau = -rV. \tag{2.27}$$

When $S \to \infty$, we make the common assumption that

$$V_{SS} \to 0. \tag{2.28}$$

Readers can refer to [20] for the details concerning this boundary condition.

To summarize, the pricing problem then consists of a set of one dimensional problems (equation (2.13)) embedded in a three dimensional (S, S_{old}, H) space. These one dimensional problems are independent between observation dates and communicate through jump conditions (2.22)-(2.23) and satisfy the boundary conditions (2.26), (2.27), (2.28) and terminal condition (2.24) with computational domain (2.25).

2.4 The Similarity Reduction

Many path-dependent option pricing problems admit similarity reductions under simplified market conditions, such as a constant volatility. When there is a similarity reduction, we can reduce the dimensionality of the problem. The existence of a similarity reduction for a given problem is closely related to the concept of homogenous functions.

Definition 1. *A function, $F(x_1, x_2, \cdots, x_n, \tau)$, is said to be homogeneous of degree m in the variables x_1, x_2, $\cdots$, x_n if:*

$$F(cx_1, cx_2, \cdots, cx_n, \tau) = c^m F(x_1, x_2, \cdots, x_n, \tau), \tag{2.29}$$

for some integer m.

For example, the standard put option payoff function at terminal time T, $P(S,\ K,\ T) = \max(K - S,\ 0)$, is homogeneous of degree one in the variables S and K since $P(cS,\ cK,\ T) = c \times \max(K - S,\ 0) = cP(S,\ K,\ T)$.

Theorem 1. *We denote $\{\chi\}$ to be the set of variables $\{y_1, y_2, \cdots, y_n\}$. Suppose that $F(x,\ \chi,\ \tau)$ satisfies the equation:*

$$F_\tau = C_1 x^2 F_{xx} + C_2 F + C_3 F_x x + C_4 x + C_5 \int_0^{+\infty} g(J) F(Jx,\ \chi,\ t) dJ, \tag{2.30}$$

where C_1, C_2, C_3, C_4, C_5 are constants. If the terminal condition $F(x,\ \chi,\ T)$ is homogeneous of degree one in variables $(x,\ \chi)$ and the boundary condition ($F_{xx} = 0$ as $x \to \infty$) is imposed, then $F(x,\ \chi,\ \tau)$ ($0 \le \tau \le T$) is homogeneous of degree one in variables $(x,\ \chi)$.

Proof. Define a new function $\hat{F}$ such that,

$$\hat{F}(\hat{x}, \hat{\chi}, \tau) = cF(x, \chi, \tau), \tag{2.31}$$

where $\hat{x} = cx; \hat{\chi} = c\chi$ and c is a constant. Then we have

$$F(x, \chi, \tau) = \frac{1}{c} \hat{F}(\hat{x}, \hat{\chi}, \tau). \tag{2.32}$$

Substitute (2.32) into (2.30) we get:

$$\begin{aligned}
\frac{1}{c}\hat{F}_\tau &= C_1 x^2 c \hat{F}_{\hat{x}\hat{x}} + C_2 \frac{1}{c}\hat{F} + C_3 x \hat{F}_{\hat{x}} + C_4 x + C_5 \frac{1}{c} \int_0^{+\infty} g(J) \hat{F}(J\hat{x}, \hat{\chi}, t) dJ \\
&= \frac{1}{c}[C_1 \hat{x}^2 \hat{F}_{\hat{x}\hat{x}} + C_2 \hat{F} + C_3 \hat{x} \hat{F}_{\hat{x}} + C_4 \hat{x} + C_5 \int_0^{+\infty} g(J) \hat{F}(J\hat{x}, \hat{\chi}, t) dJ]. \\
\Longrightarrow \hat{F}_\tau &= C_1 \hat{x}^2 \hat{F}_{\hat{x}\hat{x}} + C_2 \hat{F} + C_3 \hat{x} \hat{F}_{\hat{x}} + C_4 \hat{x} + C_5 \int_0^{+\infty} g(J) \hat{F}(J\hat{x}, \hat{\chi}, t) dJ.
\end{aligned} \tag{2.33}$$

So $\hat{F}$ also satisfies PIDE (2.30). Similarly, we assume that $\hat{F}_{\hat{x}\hat{x}} = 0$ as $\hat{x} \to \infty$. Since we suppose that the terminal condition is homogeneous of degree one, we have

$$\begin{aligned}
\hat{F}(\hat{x}, \hat{\chi}, T) &= cF(x, \chi, T) \\
&= F(cx, c\chi, T) \\
&= F(\hat{x}, \hat{\chi}, T).
\end{aligned} \tag{2.34}$$

Since $\hat{F}$ and F satisfy the same terminal condition, PIDE (2.30) and boundary conditions, we have

$$\begin{aligned}\hat{F}(\hat{x},\hat{\chi},\tau) &= F(\hat{x},\hat{\chi},\tau)\\ &= F(cx,c\chi,\tau).\end{aligned} \tag{2.35}$$

From (2.31) and (2.35) we have

$$F(cx,c\chi,\tau) = cF(x,\chi,\tau), \tag{2.36}$$

where $0 \le \tau \le T$. Thus $F(x,\chi,\tau)$ is homogeneous of degree one in the variables (x,χ). ∎

Consider the jump condition (2.22) and (2.23), if $V(S^+, S^+_{old}, H^+, t_i^+)$ is homogeneous of degree one in S, S_{old} and H, then $V(S, S_{old}, H, t_i^-)$ is homogeneous of degree one in S, S_{old} and H (since P_i is homogeneous of degree one in S, S_{old} and H). So, from Theorem 1 and equations (2.22), (2.23) we have,

$$V(cS, cS_{old}, cH, \tau) = cV(S, S_{old}, H, \tau). \tag{2.37}$$

There are at least two ways to use (2.37) to carry out a similarity reduction. If we choose, $c = \frac{H^*}{H}$, then we find that,

$$V(S, S_{old}, H, \tau) = \frac{H}{H^*} \times V(\frac{S}{H}H^*, \frac{S_{old}}{H}H^*, H^*, \tau). \tag{2.38}$$

Another possibility is to choose, $c = \frac{S^*_{old}}{S_{old}}$, then we can use the transformation,

$$V(S, S_{old}, H, \tau) = \frac{S_{old}}{S^*_{old}} \times V(\frac{S}{S_{old}}S^*_{old}, S^*_{old}, \frac{H}{S_{old}}S^*_{old}, \tau). \tag{2.39}$$

Both of these transformations only need a single value of H^* (in (2.38)) or S^*_{old} (in (2.39)) and hence reduce the dimensionality by one.

It is not clear which equation, (2.38) or (2.39), would be a better choice. To answer this question, we need to analyze the updating rule (2.23) carefully. Suppose the computational domain of S is $0 \le S \le S_{max}$, and we want to compute $V(S^*, S^*_{old}, H^*, t = 0)$ with $H^* = S^*_{old} = S^*$. For convenience, we use equation (2.38) or (2.39) to carry out the similarity reduction.

If we choose equation (2.38) to carry out the similarity reduction, then at each observation point, from equation (2.22) we have to interpolate at the point,

$$V(\frac{S^+}{H^+}H^*, \frac{S^+_{old}}{H^+}H^*, H^*, t^+). \tag{2.40}$$

Recalling equation (2.23), we have that,

$$\begin{aligned} S^+_{old} &= S^+, \\ H^+ &= \max(H, S^+), \end{aligned}$$

so that equation (2.40) becomes,

$$V\left(\frac{S^+}{\max(H, S^+)}H^*, \frac{S^+}{\max(H, S^+)}H^*, H^*, t^+\right). \tag{2.41}$$

Clearly we have

$$\frac{S^+}{\max(H, S^+)}H^* \leq H^*,$$

so that interpolated information is required only in

$$\begin{aligned} 0 \leq S \leq H^*, \\ 0 \leq S_{old} \leq H^*. \end{aligned} \tag{2.42}$$

If $H^* = S^*$, then we only require information at

$$0 \leq S_{old} \leq S^*.$$

This is very convenient since we never need information outside the original computational domain if $0 \leq S_{old} \leq S^*$.

On the other hand, if we choose equation (2.39) to carry out the interpolation, then we have to interpolate at the points,

$$V(\frac{S^+}{S^+_{old}}S^*_{old}, S^*_{old}, \frac{H^+}{S^+_{old}}S^*_{old}, t^+). \tag{2.43}$$

Recalling that $S^+_{old} = S^+$, $H^+ = \max(H, S^+)$ (from (2.23)), then equation (2.43) becomes,

$$V\left(S^*_{old}, S^*_{old}, \frac{\max(H, S^+)}{S^+} S^*_{old}, t^+\right). \tag{2.44}$$

or, recalling that $S^+ = S - P_i$, then (2.44) becomes,

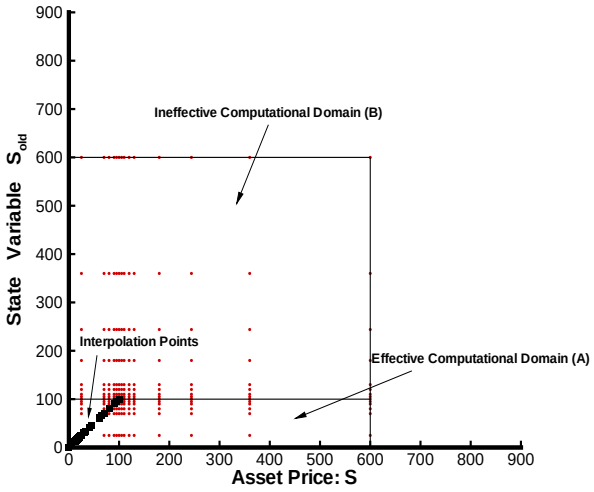
$$V\left(S^*_{old}, S^*_{old}, \frac{\max(H, S - P_i)}{S - P_i} S^*_{old}, t^+\right). \tag{2.45}$$

Then the value needed for the H interpolation, H_{inter}, is

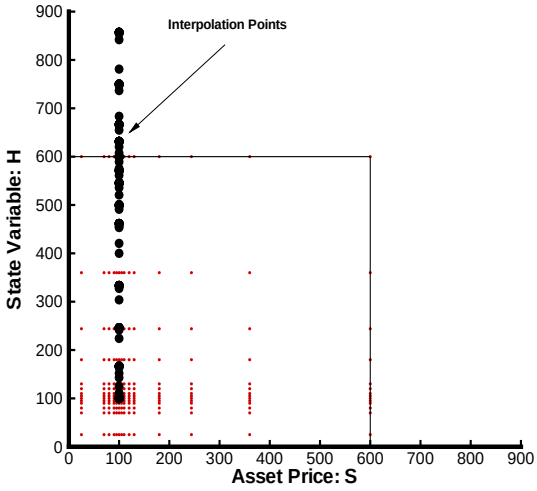
$$H_{inter} = \frac{\max(H, S - P_i)}{S - P_i} S^*_{old},$$

which can be very large if $S - P_i$ is small, and can be outside the original computational domain $0 \leq H \leq H_{max}$.

From Figure 2.1, we can see this problem more clearly. If we use identical S grids for each value of S_{old} and of H, and suppose that $S_{max} = 600$, $S^* = 100$. The "*"s in Figure 2.1 represent the interpolation points. In Figure 2.1(a), we show the interpolation situation if the equation (2.38) is used to carry out the similarity reduction. In Figure 2.1(b) we show the interpolation situation if we use (2.39) to carry out the similarity reduction. From Figure 2.1(a) we can see that the information is only required at the points $0 \leq S_{old} \leq H^* = S^*$. This means that the computational domain is restricted to a fairly small region. Conversely, in Figure 2.1(b), we can see that information is required at the points in $0 \leq H \leq H_{max}$, and for $H > H_{max}$. This is clearly undesirable.



(a)



(b)

Figure 2.1: The interpolation situation if we use (2.38) in (a), or (2.39) in (b) to carry out the similarity reduction. We assume that $S_{max} = 600; S^* = 100$. The "*"s represent interpolation points. In (a) we indicate the effective computational domain (A) and ineffective computational domain (B). Actually we do not need to compute the information at the points located in domain (B). In (b) we only draw a subset of the interpolation nodes. There are many interpolation nodes which are larger than $H = 900$.

Chapter 3

Mesh Construction

In order to solve the fundamental path-dependent pricing problem, (2.13) with the updating rule (2.22)-(2.23), we must frequently interpolate the numerical solution. The performance of the numerical method often depends crucially on the interpolation strategy and grid construction techniques. In this chapter, we discuss interpolation methods and grid construction techniques that are important to improve the efficiency of the computation.

Suppose we solve $V(S, \{\chi\}, \tau)$ where V satisfies some PIDE, (i.e. equation (2.13)) with some terminal and boundary conditions. $\{\chi\}$ represents a set of path dependent state variables. Assume that there are discrete observation points where we need to apply jump conditions. From [22] and [23], we first discretize these state variables, then for each combination of these discretized state variables, we use a numerical method to solve a PIDE. This PIDE is characterized by a set of discrete nodes $\{S_i\}$ $i = 1, 2, \cdots, i_{max}$. We solve the PIDE until we advance the solution to an observation time. At each observation time, we need to apply jump conditions, which usually require interpolation. The error in this algorithm is a result of

- The process of computing V between two adjacent observation times (the discretization error).
- The interpolation operation at each observation time.

The error from PIDE discretization is mainly affected by the S grid distribution along the asset price direction and the τ grid distribution along the time direction.

To decrease this error, we construct the computational domain so that there are more nodes in the regions where the value is changing rapidly and in the regions where we need accurate information when we apply interpolation rules.

The error from interpolation is mainly affected by the arrangement of the grid of the path dependent state variables, the method used to do the interpolation, and the accuracy of the solution value used as data for the interpolation. To decrease this error we need to analyze the updating rule and the character of the payoff and the PIDE.

To summarize, the main issues which determine solution accuracy are:

- The S grid for each PIDE solver.
- The interpolation method.
- The discretization of the auxiliary path dependent state variables.

Often these issues interact with each other.

3.1 S grid design

The S grid design is very important for computational efficiency. As a general rule, we need to add more nodes in the regions where the value changes rapidly, and in the regions where we need accurate information when we apply interpolation rules.

Recall that for path dependent contracts, we can visualize the solution method as follows. We have a set of one-dimensional PIDEs embedded in a higher dimensional space. These one-dimensional PIDEs communicate only at observation times. Consequently, each one-dimensional PIDE can have a different S grid. We need to consider how each one-dimensional PIDE, which is associated with a fixed set of path dependent state variables, interacts with the interpolation method.

In order to get the ideas across, without undue algebraic complication, we will illustrate the ideas using a simple example, a look back option [16]. Suppose we follow the numerical algorithm in [13] to price a discrete look back option. There is an auxiliary path dependent state variable M which represents the maximum value

of the asset price S up to the current time. At each observation time, M changes discretely following the updating rule,

$$M^+ = \max(S, M^-), \tag{3.1}$$

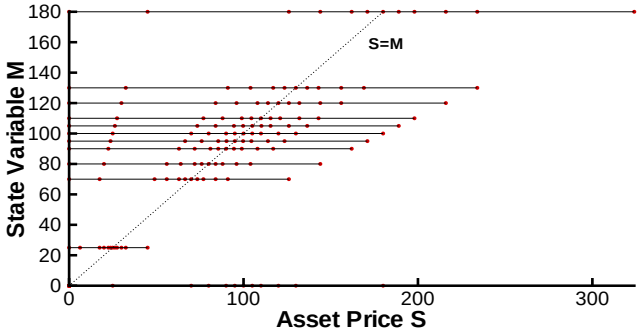
where M^- is the maximum value of S before the observation time and M^+ is the maximum value of S after the observation time. Obviously if $S > M^-$, we need to obtain values at node (S, S) in the (S, M) plane. Suppose that at time $t = 0$, $S = S^*$, then the initial value of M is also S^*. For $M = S^*$, we have a set of nodes which form a prototype S grid. Denote this set of nodes by $(S_g)^0 = (S_1)^0, \cdots, (S_{max})^0$. Normally, in the grid $(S_g)^0$, we choose a fine node spacing near $S = S^*$, since this is the region of most interest. We also assume that the grid has been constructed so that the point S^* is contained in the grid. In the following discussion, we suppose that the grid for M is equal to the grid $(S_g)^0$. Under the above preconditions, we discuss the S grids for other discrete values of M.

We consider two types of S grid design, which we refer to as the repeated grid and the scaled grid in the following. The repeated grid is such that each S grid for each different value of M is just a copy of the S grid for $M = S^*$, i.e. $(S_g)^0$. For the scaled grid, let $(S_g)^i$ represent the S grid corresponding to the discrete value M_i. The $(S_g)^i$ is composed of $(S_1)^i, \cdots, (S_{max})^i$. The following algorithm is used to construct $(S_g)^i$ $(i = 1, \cdots, i_{max})$.

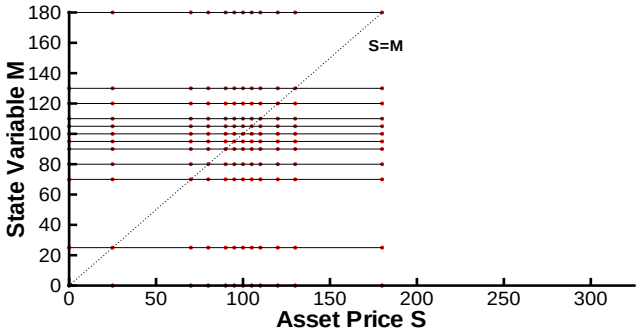
Scaled Grid Construction

For i = 1, $\cdots$, i_{max}
  For j = 1, $\cdots$, i_{max}
    $(S_j)^i = (S_j)^0 M_i / S^*$;
  EndFor
EndFor

The repeated grid and scaled grid are shown in Figure 3.1. For the scaled grid, since S^* is in the prototype grid $(S_g)^0$, for each line of constant M_i, there is a node on the diagonal $S = M$, as depicted in Figure 3.1(a). In the repeated grid, since the grid of M is equal to the prototype S grid, for each line of constant M_i, there is also a node on the diagonal $S = M$, as depicted in Figure 3.1(b).



(a) Scaled grid



(b) Repeated grid

Figure 3.1: A representative computational domain formed by discretizing the variable into a set of one dimensional problems, each of which contains a discretization of the underlying asset price.

At each observation time, if we use the repeated grid Figure 3.1(b), then no interpolation is required to enforce the jump condition (3.1) even in the case of $S > M^-$, since for all possible S values, (S, S) is always in the grid. However, the repeated grid is not generally very accurate at node $(S, M = S)$ for large S values, since usually the grid is sparse in these regions (i.e. $S \gg S^*$), which can be seen in Figure 3.1(b). We can reduce this problem by using the scaled grid in Figure 3.1(a). However, in this case, we may need interpolation to apply the jump conditions.

Clearly, there is a tradeoff between these two effects: interpolation accuracy, and accuracy of the data used to do the interpolation (the PIDE solution).

3.2 Interpolation Rule Design

In many cases, the jump conditions may require interpolation of our discrete solution. The simplest interpolation scheme is to use the value of the closest points in the computational domain to do linear or quadratic interpolation [24]. However, there are some important and special cases which allow us to do exact interpolation.

Definition 2. *An interpolation rule is termed exact if, given exact data, the interpolation produces the exact solution.*

3.2.1 Two Dimensional Case

Consider the following hypothetical example. Suppose that we have a contract with value $V = V(S, S_{old}, t)$, where S_{old} is the value of S at the previous observation. Suppose that the jump condition is

$$V(S, S_{old}, t^-) = V(S, S, t^+), \tag{3.2}$$

and suppose that

$$V(cS, cS_{old}, t) = cV(S, S_{old}, t), \tag{3.3}$$

where c is a constant. Then the jump condition (3.2) requires data at

$$V(S, S, t^+).$$

Suppose

$$S_l \leq S \leq S_h.$$

If we interpolate along the diagonal of the (S, S_{old}) plane, which is shown in Figure 3.2,

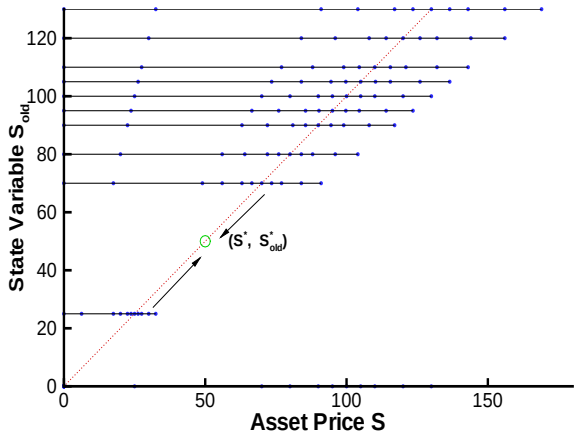


Figure 3.2: Two dimensional diagonal interpolation rule. This can be used in the case where the asset price and state variable are homogeneous.

then this is equivalent to

$$V(S, S, t^+) = \frac{S_h - S}{S_h - S_l} V(S_l, S_l, t^+) + \frac{S - S_l}{S_h - S_l} V(S_h, S_h, t^+).$$

From equation (3.3), we have

$$V(S_h, S_h, t^+) = \frac{S_h}{S_l} V(S_l, S_l, t^+),$$

then we obtain

$$V(S, S, t^+) = \frac{S}{S_l} V(S_l, S_l, t^+).$$

which is the exact result assuming $V(S_h, S_h, t^+)$ and $V(S_l, S_l, t^+)$ are exact.

In the following, we will refer to this type of interpolation as a two dimensional diagonal rule. In order for this rule to be exact, we must have $S = S_{old}$ contained in each S grid for each discrete S_{old} value.

3.2.2 Three Dimensional Case

In the case of hedge fund fees, from updating rules (2.22) and (2.23), we need values at

$$V(S^+, S^+, H^+, t^+),$$

with

$$H^+ = \max(H, S^+).$$

We would like to develop an interpolation method which is exact if the similarity reduction is valid. If $H < S^+$ then $H^+ = S^+$, assuming that the H grid is the same as the S_{old} grid, then linear interpolation along diagonal is exact. Suppose

$$S_l^+ \leq S^+ \leq S_h^+.$$

Then if we interpolate along diagonal, which is shown in Figure 3.3, then this is equivalent to

$$V(S^+, S^+, S^+, t^+) = \frac{S_h^+ - S^+}{S_h^+ - S_l^+} V(S_l^+, S_l^+, S_l^+, t^+) + \frac{S^+ - S_l^+}{S_h^+ - S_l^+} V(S_h^+, S_h^+, S_h^+, t^+).$$

Assuming that V is homogeneous of degree one in (S, S_{old}, H),

$$V(S_h^+, S_h^+, S_h^+, t^+) = \frac{S_h^+}{S_l^+} V(S_l^+, S_l^+, S_l^+, t^+).$$

Then we obtain

$$V(S^+, S^+, S^+, t^+) = \frac{S^+}{S_l^+} V(S_l^+, S_l^+, S_l^+, t^+),$$

which is the exact result assuming $V(S_h^+, S_h^+, S_h^+, t^+)$ and $V(S_l^+, S_l^+, S_l^+, t^+)$ are exact. We will refer to this type of interpolation as a three dimensional diagonal rule. In order for this rule to be exact, we must have $S = S_{old} = H$ contained in each S grid for each combination of discrete S_{old} and H values such that $S_{old} = H$.

However, if $H > S^+$ then the interpolation in the H direction is not necessary. In this case, the jump condition requires the data at

$$V(S^+, S^+, H, t^+).$$

One might suppose that the two dimensional diagonal rule would be a good rule to use in this case. Unfortunately, this is not exact in this case, since the similarity reduction (2.37) requires a scaling of all three variables(S, S_{old} and H) not just (S, S_{old}) with H fixed.

Consequently, in the case of hedge fund fees, we use equation (2.38) to carry out the similarity reduction. In the (S, S_{old}) plane, we choose a scaled (S, S_{old}) grid and use the two dimensional diagonal rule to do the interpolation, since in this way we can always use the data which is in the dense area of the S grids. This is, however, not an exact interpolation rule.

3.3 Auxiliary Path Dependent State Variable Grid Design

The grid for the path dependent state variable also affects the accuracy of interpolation and the efficiency of computation. For most path dependent options, the state variable is a function of the asset price S, so the state variable grid is highly related to the S grid design.

Consider the example of a look back option [16]. In Section 3.1, we discussed the S grid design for each value of the state variable M. For simplicity, we assume that the M grid is equal to the prototype S grid $(S_g)^0$. In fact, we should also consider the grid design of the state variable M. The design of M grid is closely related to the S grid. If we use a *repeated grid*, then we should set the M grid to be the same as the S grid. If the grids of M and S are the same, then interpolation is not

necessary. However, if we use a *scaled grid*, then we have to do interpolation for any M grid. If the look back option satisfies the similarity reduction in variables S and M, then the interpolation is exact. However, even if a similarity reduction is not valid, the *scaled grid* with two dimensional diagonal interpolation rule still works well [18].

In the case of hedge fund fees, if the similarity reduction is not valid, then from the jump condition (2.22) and (2.23), we require data at

$$V(S^+, S^+_{old}, H^+, t^+_i),$$

with

$$\begin{aligned} S^+_{old} &= S^+, \\ H^+ &= \max(H, S^+). \end{aligned}$$

Obviously we have $H^+ \geq S^+_{old}$ in $V(S^+, S^+_{old}, H^+, t^+_i)$, which means that in each plane of H, we can limit the grid of S_{old} to $H \geq S_{old} \geq 0$, which is usually much smaller than the area $S_{max} \geq S_{old} \geq 0$.

If $V(S, S_{old}, H, t)$ admits a similarity reduction (2.37) ($V(cS, cS_{old}, cH, \tau) = cV(S, S_{old}, H, \tau)$), then instead of solving $V(S, S_{old}, H, t)$ directly, we always solve $\frac{H}{H^*}V(\frac{S}{H}H^*, \frac{S_{old}}{H}H^*, H^*, t)$ for fixed H^*. Then from equation (2.42) ($0 \leq S \leq H^*, 0 \leq S_{old} \leq H^*$), we can limit the grid of S_{old} to $S^* \geq S_{old} \geq 0$ if H^* equal to S^*. We only need a rectangular computational domain, as shown in Figure 2.1(a) (domain (A)), to solve the problem, which can greatly improve the efficiency of computation.

From the above discussion, we can see that the three issues(S grid design, interpolation rule design and path dependent state variable grid design) impact each other. There is not a rule which is better than others in all different cases. Generally, when we consider the solution of a path dependent option pricing problem, we need to

- Check whether the partial differential equation and jump conditions satisfy a similarity reduction (Section 2.4). If possible, we should, of course, use the similarity reduction. Using a similarity reduction can not only reduce the dimensionality of the problem but also impact the interpolation rule design.
- Design the interpolation rule. If the problem satisfies a similarity reduction, we should use a two dimensional diagonal rule or three dimensional diagonal

rule to get exact interpolation. Even in the case where a similarity reduction is not valid, these interpolation rules usually work well [18].

- Design the path dependent state variables grids using the interpolation rule and jump conditions. Information which is unnecessary when applying the interpolation rule should be taken into account when designing the grid.
- Design one-dimensional PIDE grids for each combination of path dependent state variables. In this process, we should put more nodes in the regions where the value changes rapidly, and in regions where we need accurate information when we do the interpolation.
- Put all these criteria together to check whether they are compatible.

In addition, in our case, in light of the boundary condition (2.26), we always put a node on the grid $(b \times H, S_{old}, H)$ [33], corresponding to the barrier node.

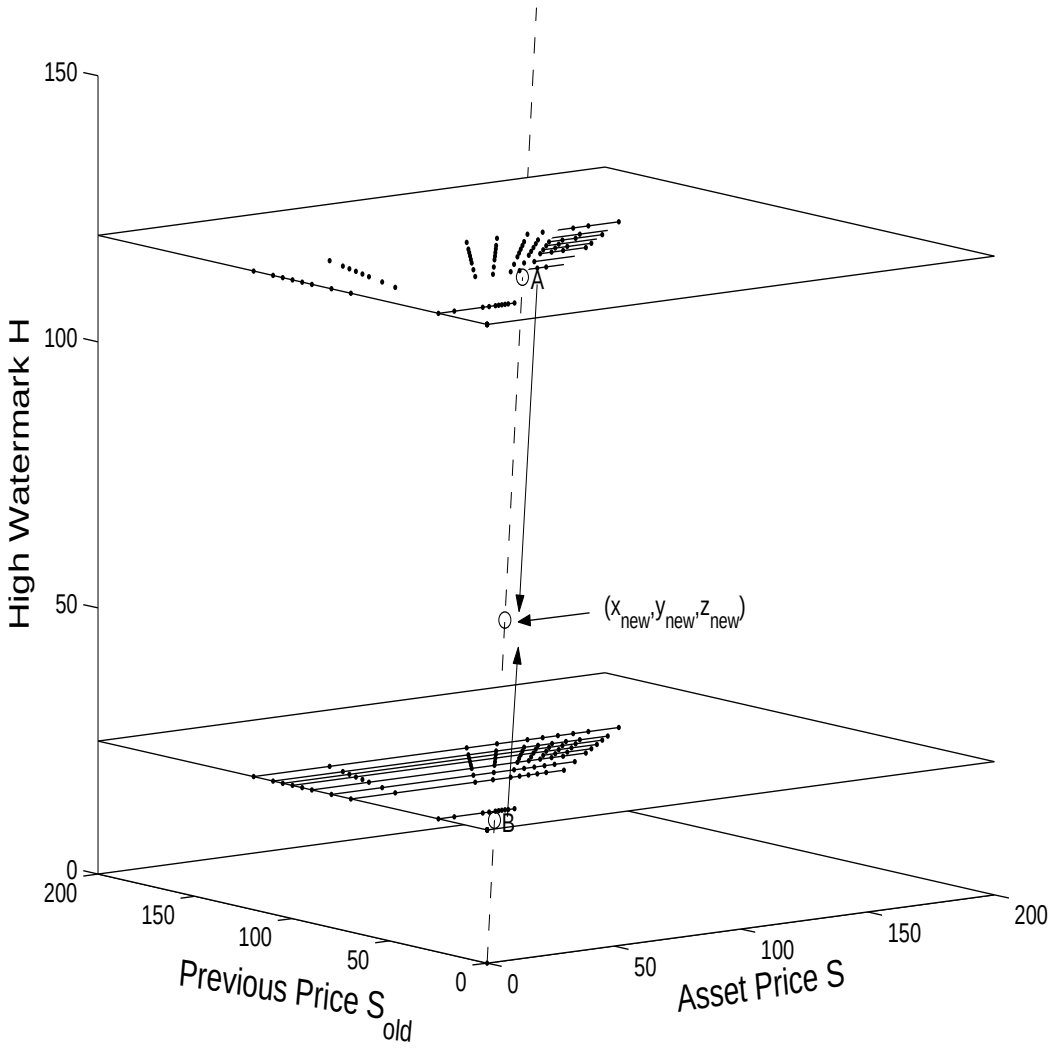


Figure 3.3: Three dimensional diagonal rule: notice that we still need to do inexact interpolation for point A and point B, which may not be located on the computational grid.

Chapter 4

PIDE Discretization

In this chapter, we give a brief overview of the method used to solve (2.13). Readers can refer [25, 19, 41] for more details. In general, we use finite difference to discretize the PIDE, use Fast Fourier Transform(FFT) method to compute the integral term and use a fixed point iteration technique to obtain the performance fee value at time τ^{n+1} from the value at time τ^n.

4.1 Finite Difference

In this section, we will derive the discretized equation from equation (2.13) using the finite difference method. First, Define a grid of points in the (S, τ) plane:

$$\begin{aligned} S_0, S_1, \cdots, S_m \qquad & 0 \leq S_i \leq S_m, \\ \tau_n = n\Delta\tau \qquad & 0 \leq \tau_n \leq N\Delta\tau = T. \end{aligned}$$

Then let

$$V(S_i, \tau_n) = V_i^n.$$

The time derivative in equation (2.13) is approximated by

$$\left(\frac{\partial V}{\partial \tau}\right)_i^n \simeq \frac{V_i^{n+1} - V_i^n}{\delta\tau}. \tag{4.1}$$

The second order derivative term in equation (2.13) is approximated by

$$\left(\frac{1}{2}\sigma^2 S^2 V_{SS}\right)_i^n \simeq \frac{1}{2}\sigma^2 S_i^2 \left(\frac{\left(\frac{V_{i+1}^n - V_i^n}{S_{i+1} - Si}\right) - \left(\frac{V_i^n - V_{i-1}^n}{S_i - S_{i-1}}\right)}{\frac{S_{i+1} - S_{i-1}}{2}} \right). \tag{4.2}$$

The discounting term in equation (2.13) is represented by

$$[(r+\lambda)V]_i^n = (r+\lambda)V_i^n. \tag{4.3}$$

The first derivative term on the right hand side of equation (2.13) can be approximated in three ways.
A central difference:

$$((r - m_{total} - \kappa\lambda)SV_S)_i^n \simeq (r - m_{total} - \kappa\lambda)S_i \left(\frac{V_{i+1}^n - V_{i-1}^n}{S_{i+1} - S_{i-1}}\right). \tag{4.4}$$

A forward difference:

$$((r - m_{total} - \kappa\lambda)SV_S)_i^n \simeq (r - m_{total} - \kappa\lambda)S_i \left(\frac{V_{i+1}^n - V_i^n}{S_{i+1} - S_i}\right) \tag{4.5}$$

A backward difference:

$$((r - m_{total} - \kappa\lambda)SV_S)_i^n \simeq (r - m_{total} - \kappa\lambda)S_i \left(\frac{V_i^n - V_{i-1}^n}{S_i - S_{i-1}}\right). \tag{4.6}$$

If $S_{i+1} - S_i = S_i - S_{i-1}$, then equation (4.4) is second order, while equation (4.5) and (4.6) is only first order. Consequently, we will want to use the central approximation as mush as possible. Substituting equations (4.1), (4.2), (4.3) and one of (4.4), (4.5) or (4.6) into equation (2.13) gives a discrete equation of form,

$$V_i^{n+1} = V_i^n(1 - (\alpha_i + \beta_i + r + \lambda)\Delta\tau) + V_{i-1}^n \Delta\tau\alpha_i + V_{i+1}^n \Delta\tau\beta_i \tag{4.7}$$

$$+ m_c S_i \Delta\tau + \Delta\tau\lambda \int_0^{+\infty} g(J)V(JS,t)dJ, \tag{4.8}$$

where if we discretize the first derivative term of (2.13) with central differences

(4.4), this results in

$$\begin{aligned}\alpha_{i,central} &= \frac{\sigma_i^2 S_i^2}{(S_i - S_{i-1})(S_{i+1} - S_{i-1})} - \frac{(r - \lambda\kappa - m_{total})S_i}{S_{i+1} - S_{i-1}} \\ \beta_{i,central} &= \frac{\sigma_i^2 S_i^2}{(S_{i+1} - S_i)(S_{i+1} - S_{i-1})} + \frac{(r - \lambda\kappa - m_{total})S_i}{S_{i+1} - S_{i-1}}.\end{aligned} \tag{4.9}$$

If $\alpha_{i,central}$ or $\beta_{i,central}$ is negative, oscillations may appear in the numerical solution. These can be avoided by using forward differences (4.5) at the problem nodes, leading to (forward difference)

$$\begin{aligned}\alpha_{i,forward} &= \frac{\sigma_i^2 S_i^2}{(S_i - S_{i-1})(S_{i+1} - S_{i-1})} \\ \beta_{i,forward} &= \frac{\sigma_i^2 S_i^2}{(S_{i+1} - S_i)(S_{i+1} - S_{i-1})} + \frac{(r - \lambda\kappa - m_{total})S_i}{S_{i+1} - S_{i-1}}.\end{aligned} \tag{4.10}$$

4.2 Fast Fourier Transform(FFT)

The integral term $I(S) = \int_0^{+\infty} g(J)V(JS,t)dJ$ in (2.13) can be transformed to

$$I(S) = \int_{-\infty}^{+\infty} \overline{V}(x+y)\overline{f}(y)dy, \tag{4.11}$$

where $\overline{V}(x) = V(e^x)$, $\overline{f}(y) = g(e^y)e^y$. The integral (4.11) can be evaluated efficiently using an FFT. In the FFT, an equally spaced grid in $\log S$ coordinates is needed, which may not coincide with an unequally spaced grid in S coordinates. So we linearly interpolate to get $\overline{V}_j$ from V_k, i.e. if

$$S_{\gamma(j)} \le e^{j\Delta x} \le S_{\gamma(j)+1},$$

then

$$\overline{V}_j = \psi_{\gamma(j)} V_{\gamma(j)} + (1 - \psi_{\gamma(j)})V_{\gamma(j)+1} + O((\Delta S_{\gamma(j)+1/2})^2), \tag{4.12}$$

where $\psi_{\gamma(j)}$ is an interpolation weight, and $\Delta S_{i+1/2} = S_{i+1} - S_i$.

After obtaining $I(S_i) = I_i$ using the FFT, we need to interpolate back to the S grid, i.e. if

$$e^{x_{\pi(k)}} \le S_k \le e^{x_{\pi(k)+1}},$$

then

$$I(S_k) = \phi_{\pi(k)} I_{\pi(k)} + (1 - \phi_{\pi(k)}) I_{\pi(k)+1} + O((e^{x_{\pi(k)}} - e^{x_{\pi(k)+1}})^2), \tag{4.13}$$

where $\phi_{\pi(k)}$ is an interpolation weight. Note that

$$0 \leq \phi_i \leq 1, \quad 0 \leq \psi_i \leq 1.$$

Then we have

$$I(S_k) = \sum_{j=-\frac{N}{2}+1}^{j=\frac{N}{2}} \chi(V, k, j) \overline{f}_j \Delta y, \tag{4.14}$$

where $V = [V_0, V_1, \cdots V_m]^T$ and $m+1$ is the number of nodes in S coordinates, $\overline{f}_j = g(e^{j\Delta y}) e^{j\Delta y}$ and

$$\begin{aligned} \chi(V, k, j) = \phi_{\pi(k)} [\psi_{\gamma(\pi(k)+j)} V_{\gamma(\pi(k)+j)} + (1 - \psi_{\gamma(\pi(k)+j)}) V_{\gamma(\pi(k)+j)+1}] + \\ (1 - \phi_{\pi(k)}) [\psi_{\gamma(\pi(k)+1+j)} V_{\gamma(\pi(k)+1+j)} + (1 - \psi_{\gamma(\pi(k)+1+j)}) V_{\gamma(\pi(k)+1+j)+1}]. \end{aligned} \tag{4.15}$$

Another issue is that the FFT effectively assumes that the input functions are periodic. This may cause wrap-around pollution unless special care is taken when implementing the algorithm. In this thesis, we use the technique suggested in [41] to solve this problem.

4.3 Fixed Point Iteration

After evaluating the integral term, we use a fixed point iteration method to get the value at time τ^{n+1} from the value at time τ^n. Define a $(m+1) \times (m+1)$ matrix $\hat{M}$ such that

$$\hat{M} = \Delta\tau \begin{pmatrix} -(r+\lambda) & 0 & & & \\ \alpha_1 & -(\alpha_1 + \beta_1 + r + \lambda) & \beta_1 & & \\ & & \ddots & & \\ & & \alpha_{m-1} & -(\alpha_{m-1} + \beta_{m-1} + r + \lambda) & \beta_{m-1} \\ & & \gamma_{m-2} & \gamma_{m-1} & \gamma_m \end{pmatrix}$$

From boundary assumption $V_{SS} \to 0$ when $S \to \infty$ and backward difference (4.6), we have [20],

$$\begin{aligned}\gamma_{m-2} &= 0\\ \gamma_{m-1} &= -\frac{S_m(r - m_{total} - \kappa\lambda)}{S_m - S_{m-1}}\\ \gamma_m &= -(r+\lambda) - \gamma_{m-1}\end{aligned}$$

Also define the vector $\Omega(V^n)$ (which is a linear function of V^n) such that

$$[\Omega(V^n)]_i = \sum_{j=-\frac{N}{2}+1}^{j=\frac{N}{2}} \chi(V^n, i, j)\overline{f}_j \Delta y. \tag{4.16}$$

Thus we can write a fully implicit($\theta = 0$) or Crank-Nicolson($\theta = \frac{1}{2}$) discretization as

$$[I - (1-\theta)\hat{M}]V^{n+1} = [I + \theta\hat{M}]V^n + (1-\theta)\lambda\Delta\tau\Omega(V^{n+1}) + \theta\lambda\Delta\tau\Omega(V^n) + Im_c S\Delta\tau, \tag{4.17}$$

where I is a $m+1$ dimensional identity matrix, and $S = [S_0, S_1, \cdots S_m]^T$.
We can then derive the fixed point iteration method [19, 41] as follows:

Fixed point Iteration

Let $(V^{n+1})^0 = V^n$
Let $\hat{V}^k = (V^{n+1})^k$
For k = 0,1,2,$\cdots$ until convergence
  Solve $[I - (1-\theta)\hat{M}]\hat{V}^{k+1} = [I + \theta\hat{M}]V^n + (1-\theta)\lambda\Delta\tau\Omega(\hat{V}^k) + \theta\lambda\Delta\tau\Omega(V^n) + Im_c S\Delta\tau$
  If $\max_i \frac{|\hat{V}_i^{k+1} - \hat{V}_i^k|}{\max(1, |\hat{V}_i^{k+1}|)} <$ tolerance then quit
EndFor

Chapter 5

Valuation Using the Monte Carlo Approach

The Monte Carlo (MC) approach is a statistical technique that simulates a stochastic process directly rather than solving the underlying differential equations that describe the pricing problem.

The basic steps involved in the MC method are:

- Simulate a geometric Brownian motion sample path with jump diffusion under risk neutral measure.
- Compute the payoff produced by the sample path and discount it to time zero.
- Add the discounted payoffs from many sample paths together, then compute the arithmetic average of these payoffs.

Typically we use forward Euler to integrate the stochastic differential equation to produce each sample path. We refer to this method as a timestepping Monte Carlo (MC) method in the following. There is, however, a problem with this approach. The time discretization of the jump diffusion paths introduces a timestepping bias in the pricing estimate. This bias should not be underestimated. To reduce the bias, one should make the time discretization fine enough, but as a result the computation becomes very slow. In this chapter, we provide an algorithm which

eliminates the timestepping bias completely to value the option-like performance fee. In addition, this method is orders of magnitude more efficient computationally than the timestepping MC method.

We start by generating the jump times of the process. The time intervals between any two adjacent jump events are independent and identically distributed exponential random variables with mean $1/\lambda$. Since the jump size follows an independent lognormal distribution, we can obtain the asset value immediately before and immediately after the jump instant given the time interval and jump size. Then, between these generated points, we have a pure diffusion with known end points, hence a Brownian bridge [34].

We assume that investors withdraw their money without any cost, if the asset price drops below some given level (the liquidation barrier). Once this situation occurs, there is no future performance fee. This performance fee is similar to a continuously observed knock out barrier option. Using the Brownian bridge concept, we can obtain the barrier-crossing probability in this interval between two adjacent jump events, given the two end-points of the pure diffusion process. With this probability, we can generate a uniform random variable to decide if the barrier is crossed in this interval [35]. Note that we do not use forward Euler to integrate the stochastic differential equation, so there is no timestepping error. The detailed algorithm is shown in Appendix A.

5.1 Formulation

To simulate a sample path with a jump diffusion, we first determine two things, the jump event time and jump size.

Recall that we assume that the jump size J follows the lognormal distribution (i.e. $J \sim LN(\mu, \gamma)$) and that the distribution of J is independent of dq and dz. Consequently, by generating a lognormally distributed random number, we can determine the jump size.

For the jump event time, we can simulate the sequence of interval times between two adjacent jump events $\Delta t_j^*, j = 1, 2, 3, \cdots$, which are independent identically distributed exponential random variables with mean $1/\lambda$.

After obtaining the sequence of intervals between jump events we can get the jump event times $0 < t_1^* < t_2^* < \cdots$ [21], where $t_j^*, j = 0, 1, 2, \ldots$ is the j^{th} random arrival time of the jump event. Since we assume that dz and dq in (2.1) are independent of each other, $S(t)$ evolves as an ordinary geometric Brownian motion from one jump time to the next. When a jump occurs, $S(t)$ changes to $J(t)S(t)$. To be more precise, under the risk neutral measure, equation (2.1) can be re-written in the following form:

$$\begin{aligned}\frac{dS}{S} &= (r - \lambda\kappa - m_{total})dt + \sigma dz, && \text{if a jump does not happen,}\\ &= (r - \lambda\kappa - m_{total})dt + \sigma dz + (J-1), && \text{if a jump happens.}\end{aligned}$$

Let t_j^{*-} denote the instant before the j^{th} jump event occurs, let t_j^{*+} denote the instant after the j^{th} jump event, and let $\Delta t_{j+1}^* = t_{j+1}^* - t_j^*$ denote the time interval between the $(j+1)^{th}$ and the j^{th} jump events. Then we have the following exact solution of $S(t)$:

$$\begin{aligned}S(t_{j+1}^{*-}) &= S(t_j^{*+}) \exp[(r - m_{total} - \lambda\kappa - \frac{\sigma^2}{2})\Delta t_{j+1}^* + \sigma\phi\sqrt{\Delta t_{j+1}^*}]\\ S(t_{j+1}^{*+}) &= S(t_{j+1}^{*-})J(t_{j+1}),\end{aligned} \tag{5.1}$$

where ϕ is a random variable drawn from a standard normal distribution.

The process follows a pure Brownian motion between any two adjacent jump times t_j^{*+} and t_{j+1}^{*-}. Since we know the initial and terminal values (from equation (5.1)), the Brownian motion is actually a Brownian bridge [34].

Let B_t^j be a Brownian bridge in the interval $[t_{j-1}^{*+}, t_j^{*-}]$ with two end points value $B^j_{t_{j-1}^{*+}} = S(t_{j-1}^{*+})$, $B^j_{t_j^{*-}} = S(t_j^{*-})$, and $\tau_j = t_j^{*-} - t_{j-1}^{*+}$. From [34], we can obtain the probability that the minimum of B_t^j is always above the barrier in the interval τ:

$$\begin{aligned}P_j &= P\left(\inf_{t_{j-1}^{*+} \le t \le t_j^{*-}} (B_t^j) > H | B^j_{t_{j-1}^{*+}} = S(t_{j-1}^{*+}), B^j_{t_j^{*-}} = S(t_j^{*-})\right)\\ &= \begin{cases} 1 - \exp(-\frac{2[\log H - \log S(t_{j-1}^{*+})][\log H - \log S(t_j^{*-})]}{\tau_j \sigma^2}) & \text{if} \quad \min[S(t_{j-1}^{*+}), S(t_j^{*-})] > H,\\ \\ 0 & \text{otherwise,}\end{cases}\end{aligned} \tag{5.2}$$

where H is the barrier level, and σ is the volatility.

5.2 Convergence and Error Estimation

There are two sources of error in the timestepping MC approach: timestepping error and sampling error. In the timestepping MC method, forward Euler generates the timestepping error. The timestepping error is of size $O(\Delta t)$ [36, 37]. The source of the sampling error lies in using the MC method to calculate the mean. The sampling error is of size $O(1/\sqrt{M})$, when M is the number of sample paths.

As we noted above, the algorithm in Appendix A can eliminate the timestepping error completely, so we need only to consider the sampling error. Let V^m be the present value of the sum of the payoffs in the mth path, where $m = 1, \cdots, M$. The estimated mean of the sample is given by

$$\hat{\mu} = \frac{1}{M} \sum_{m=1}^{M} (V^m), \tag{5.3}$$

and the standard deviation of the estimate is given by

$$\omega = [\frac{1}{M-1} \sum_{m=1}^{M} (V^m - \hat{\mu})^2]^{\frac{1}{2}}. \tag{5.4}$$

Then the 95% confidence interval for the actual value V of the performance fee is ([41, 42])

$$\hat{\mu} - \frac{1.96\omega}{\sqrt{M}} \leq V \leq \hat{\mu} + \frac{1.96\omega}{\sqrt{M}}. \tag{5.5}$$

Note that in order to reduce this sampling error by a factor of 2, the number of simulations must be increased by 4 [36].

Chapter 6

Test Results

In this chapter, we will present numerical tests for the PIDE and MC algorithm for valuing the hedge fund performance fee. We will look at the convergence rate of the PIDE methods, as well as a comparison of numerical results using the two methods. In order to evaluate the role of investor's withdrawal option, we will show the results under two assumptions. The first case assumes that the investors withdraw their money without any cost when the value of their investment drops below half of high-water mark (equation (2.26), $b = 0.5$). The second case assumes that the investors are not allowed to withdraw their money until the contract expires (equation (2.26), no-barrier).

6.1 Contract Details and Input Parameters

All numerical tests of the PIDEs use Crank-Nicolson with the Rannacher smoothing technique [38, 39], the diagonal interpolation rule (Section 3.2) and constant time steps. Details of the contract used in these tests are provided in Table 6.1. In these tests, we use a constant volatility model with jump diffusion. Input parameters are presented in Table 6.2.

6.2 Convergence Rate Tests

In this section, we test the convergence rate of our numerical algorithm for the PIDE (2.13) as the grid size is refined. Since the volatility is assumed constant, we can use the similarity reduction to reduce this problem to a two dimensional PIDE.

Parameter	Value
Observation times	1.0,2.0,3.0,4.0,5.0
T	5.0
Performance rate	0.2
Hurdle rate	0.05
Management rate m_{total}	0.01
m_c	0

Table 6.1: Hedge fund contract details.

Parameter	Value
S^*	100
σ	0.25
r	0.05
λ	0.1
μ	-0.9
γ	0.45

Table 6.2: Parameters for the constant volatility case with jumps.

A series of tests were carried out, where at each refinement level, new nodes were inserted between each pair of nodes in the coarser grid, and the timestep is halved. The results of the convergence tests are shown in Table 6.3. From the results, convergence appears to be asymptotically quadratic. If we do not use the similarity reduction, then we have to solve a three dimensional problem and the results are shown in Table 6.4. Compared with the values in Table 6.3, we can see that they are in good agreement. We can also note that in Table 6.4, the convergence rate is only first order. This is probably due to the fact that the three dimensional interpolation is not exact.

Refine-ment	Number of grid nodes in S, S_{old} direction	Time steps	Fixed point iterations	Value	Difference	Ratio
Crank-Nicolson ($b = 0.5$)						
0	35	40	114	8.836762424		
1	69	80	225	8.91300492	0.076242496	
2	137	160	449	8.931048657	0.018043737	4.23
3	273	320	685	8.935510431	0.004461774	4.04
4	545	640	1227	8.936638217	0.001127786	3.96
Crank-Nicolson (no-barrier)						
0	35	40	120	8.942057159		
1	69	80	234	8.995195025	0.053137866	
2	137	160	463	9.006712117	0.011517092	4.61
3	273	320	745	9.009408524	0.002696407	4.27
4	545	640	1259	9.010072595	0.000664071	4.06

Table 6.3: The value transferred from the investors to the managers of hedge funds. Contract details are provided in Table 6.1. Market parameters are given in Table 6.2. A similarity reduction is used, so no grid is needed in the H direction. At each refinement level, new nodes are inserted between each pair of grid nodes on the coarser grid, and the timestep is halved. Difference refers to the change in numerical value from one level of refinement to the next. Ratio refers to the ratio of difference between successive refinements.

6.3 Monte Carlo Test Results

In this section, we use the MC method to obtain the value of the option-like performance fee. We use the contract and input parameters given in Section 6.1. To

Refine-ment	Number of grid nodes in S, S_{old}, H direction	Time steps	Fixed point iterations	Value	Difference	Ratio
Crank-Nicolson ($b = 0.5$)						
0	35	40	97	8.826223921		
1	69	80	196	8.899703693	0.073479772	
2	137	160	391	8.926223287	0.026519594	2.77
3	273	320	613	8.933267855	0.007044568	3.76
4	545	640	1108	8.936269911	0.003002056	2.35
Crank-Nicolson (no-barrier)						
0	35	40	117	8.933938289		
1	69	80	226	8.98196105	0.048022761	
2	137	160	438	9.00193398	0.01997293	2.40
3	273	320	730	9.007176061	0.005242081	3.81
4	545	640	1223	9.009716259	0.002540198	2.06

Table 6.4: The value transferred from the investors to the managers of hedge funds. Contract details are provided in Table 6.1. Market parameters are given in Table 6.2. A full three dimensional PIDE is solved. At each refinement level, new nodes are inserted between each pair of grid nodes on the coarser grid, and the timestep is halved. Difference refers to the change in numerical value from one level of refinement to the next. Ratio refers to the ratio of difference between successive refinements.

eliminate the timestepping error, we use the algorithm in Appendix A to generate the sample path for the case $b = 0.5$. For the no-barrier case, since the investors are not allowed to withdraw their money, we do not need to compute the probability of crossing the barrier. As the timestepping error has been eliminated, we only need to consider the sampling error of the MC method.

To analyze the sampling error, we compute the mean $\hat{\mu}$ and the standard error $\frac{\omega}{\sqrt{M}}$, where ω is the standard deviation and M is the number of sample paths. The results are shown in Table 6.5. From Table 6.5, we can observe that the standard sampling error is linear to $1/\sqrt{M}$.

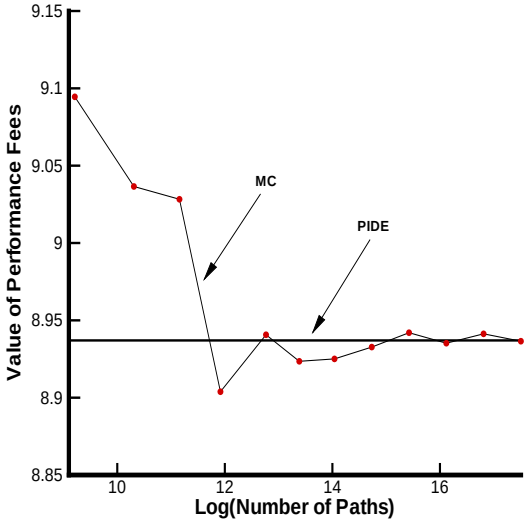
Number of Sample Paths (M)	$1/\sqrt{M}$	Value ($b = 0.5$)	Standard Error ($b = 0.5$)	Value (no-barrier)	Standard Error (no-barrier)
10,000	1/100	9.08239	0.114064	8.90413	0.11122
40,000	1/200	9.02045	0.0572686	9.0809	0.0568983
160,000	1/400	8.91238	0.0281759	9.00866	0.028335
640,000	1/800	8.95142	0.0141302	9.00977	0.0141273
2,560,000	1/1600	8.92982	0.00705891	9.00864	0.0070529
10,240,000	1/3200	8.93318	0.00353282	9.00771	0.00352613
40,960,000	1/6400	8.93647	0.00176678	9.01276	0.00176398
163,840,000	1/12800	8.93688	0.000883206	9.00999	0.0008818

Table 6.5: This table shows the standard error for the MC algorithm using parameters in Table 6.1 and 6.2.

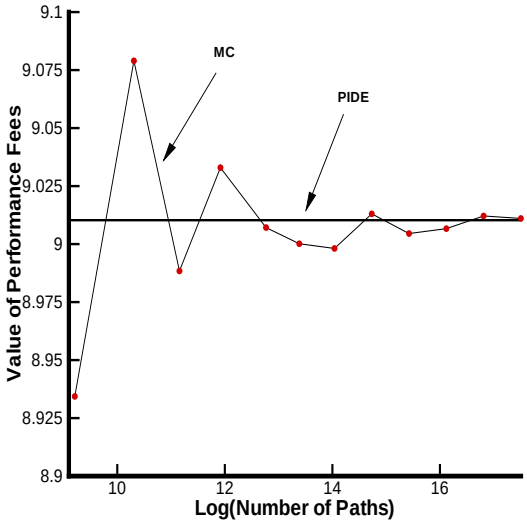
Using the equation (5.5), and substituting in values from Table 6.5, we obtain

$$8.93515 \leq V \leq 8.93861 \qquad \text{For the case } b = 0.5$$
$$9.00826 \leq V \leq 9.01172 \qquad \text{For the case no-barrier}$$

This implies that in the case $b = 0.5$, we are 95% confident that the true value of the performance fees lies between \$8.93515 and \$8.93861. In the no-barrier case, we are 95% confident that the true value of the performance fees lies between \$9.00826 and \$9.01172. In each case, the value of numerical methods with the highest refinement level lies inside the 95% confidence interval of the MC estimates (from Table 6.3



a. $b = 0.5$

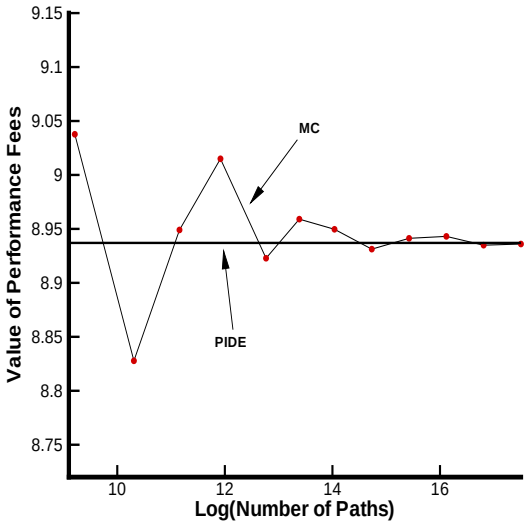


b. no-barrier

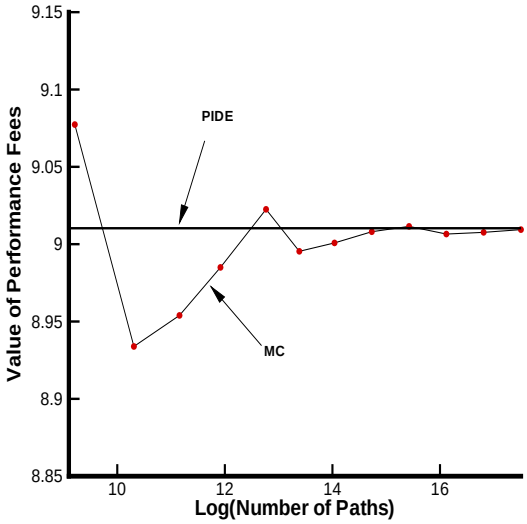
Figure 6.1: This figure shows the convergence of the value of the performance fees using the Monte Carlo method which uses the algorithm in Appendix A to produce a sample path. Contract parameters are given in Table 6.1 and market parameters are given in Table 6.2.

and Table 6.4). Figure 6.1 plots the convergence of the MC method to the PIDE solution for both case $b = 0.5$ and no-barrier case.

We then use the timestepping MC method and check the barrier condition at each time step to determine the value of the performance fees. The test results are shown in Figure 6.2. In Figure 6.2, the time step decreases from one month to one day as the number of sample paths increases from $10,000$ to $40,000,000$. From Figure 6.2, we can observe that the timestepping MC method also converges to the PIDE solution. However, on an SGI machine with SUSE Linux (kernel version: 2.6.5-7.244-sn2), when the number of sample paths is equal to $40,000,000$, the algorithm in Appendix A (using the Brownian bridge) only requires 152.3 seconds for the case $b = 0.5$ and 87.5 seconds for the no-barrier case. The corresponding CPU times using the timestepping MC method are 11646.2 seconds (the case $b = 0.5$) and 14640.7 seconds (the no-barrier case). We can see that the algorithm using the Brownian bridge is much more efficient than the timestepping MC method.



a. $b = 0.5$



b. no-barrier

Figure 6.2: This figure shows the convergence of the value of the performance fees using the timestepping Monte Carlo method. Contract parameters are given in Table 6.1 and market parameters are given in Table 6.2. The time step decreases from one month to one day as the number of sample paths increases from $10,000$ to $40,000,000$.

Chapter 7

Interpretation of the Model

In this chapter, we will use the model (2.13) to analyze the hedge fund performance fee. Note that in the following, we suppose that the management rate $m_{total} = 0.01$, the extra compensation rate $m_c = 0$ and the initial value $S^* = 100$. Then we have the following parameters which can affect the present value of the performance fees:

σ	is the volatility.
λ	is the mean arrival rate of the Poisson process.
μ	is the mean of log(J), where J is jump size.
γ	is the standard deviation of log(J), where J is jump size.
b	is the liquidation level which is a constant fraction of the high-water mark (refer to Section 2.2).
T	is the lifetime of the hedge fund contract.
p	is the performance rate.
h	is the hurdle rate.
S^*	is the initial asset price.

σ, λ, μ and γ are market parameters; b is an endogenous choice made by the investor; T, p and h are contractual parameters. In the following sections, we will analyze the effects of these parameters on the value of the hedge fund performance fees. Except where otherwise specified, the test parameters are $b = 0.5$, $\sigma = 0.25$, $\lambda = 0.1$, $\mu = -0.9$, $\gamma = 0.45$, $T = 5.0$, $p = 0.2$, $h = 0.05$, $S^* = 100$.

7.1 The Effect of the Endogenous Choice for b

Since hitting the liquidation barrier cancels all future fee payments, a barrier near S^* reduces the value of future fees. For the same reason, the value of the performance fees decreases as b increases, which is shown in Figure 7.1. From Figure 7.1, we can see that when b is small, the influence of withdrawal is small. However, when b is more than 0.60 (i.e. the barrier is close to the asset price), withdrawal reduces the value of the performance fees by a large amount.

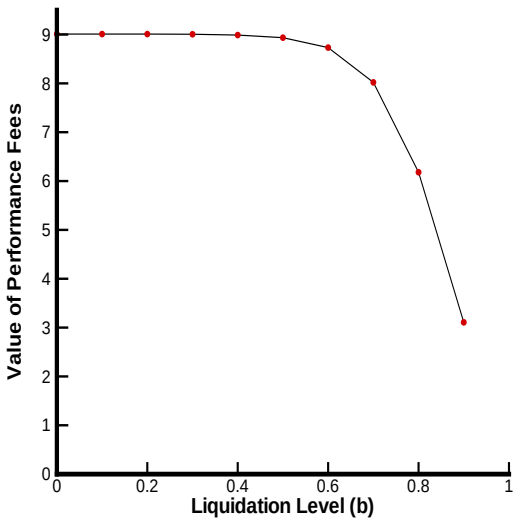


Figure 7.1: This figure shows the effect of b on the value of the performance fee.

7.2 The Effect of Market Parameters

The present value of the performance fees is generally increasing as σ and λ increase, since the performance fee has option-like characteristics. An exception occurs when the asset value is close to the liquidation barrier. This is similar to the well-known result that the value of a knock-out barrier option can be decreasing in volatility if the underlying asset price is close to the barrier. This is because an increase in σ increases the possibility of hitting the liquidation barrier. If the barrier is

close to the asset price, this effect dominates, and the value of the performance fees decreases (see Figure 7.2 (a) line B).

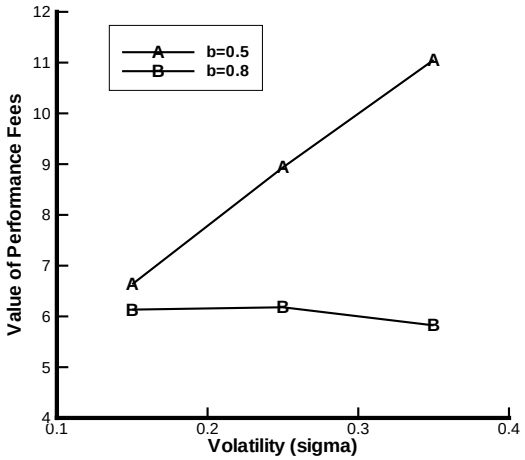
The mean jump arrival rate λ generally has a similar effect to the volatility σ when the barrier is far away from the asset price. However increases in σ and λ have different effects on the value of the performance fees when the barrier is close to the asset price. As σ increases, the asset price fluctuates more intensely, but the extent is much smaller than the fluctuation from jump events. When a jump event occurs, the asset price may increase or decrease by a large amount suddenly. If the asset price decreases suddenly and hits the liquidation barrier, the future performance fees become zero. If the asset price increases suddenly, the performance fees increases by a large amount. Note that the loss is limited in magnitude (no future fees at most), but there is no upper limit of increase. Thus the effects of increase and decrease are not symmetric. The effect of increase dominates the effect of decrease when λ increases (see Figure 7.2 (b) line B).

However, when σ increases, the asset fluctuations are not as extreme as in the jump case. If the barrier is close to the current asset price, the effect of hitting the barrier dominates the increase in the asset price. That is why the effect of the increase in σ and λ on the value of the performance fees is different if the liquidation barrier is close. From Figure 7.2, we can see the difference clearly.

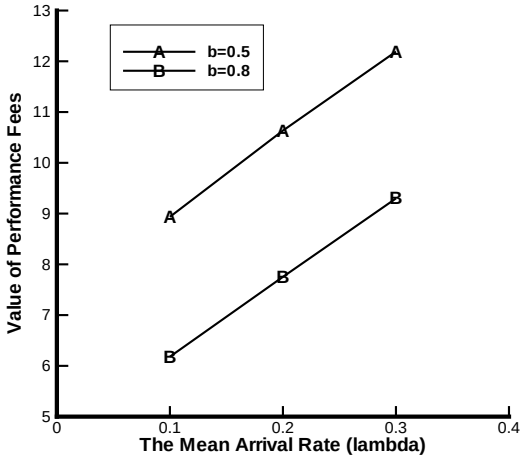
From Figure 7.2, we can observe that if b is not close to one, the volatility σ and the mean jump arrival rate λ have a similar effect on the value of hedge fund performance fees. If we assume that there is no jump diffusion, we can calibrate the volatility σ to the value of the performance fees with jump diffusion. In other words, we can adjust σ in a model with $\lambda = 0$, to obtain the same value for the performance fees as in the full jump diffusion model. We will refer to this calibrated volatility as the implied volatility in the following.

Table 7.1 shows the implied volatility (no jump) which gives the same fee value as in the $\lambda \neq 0$ jump model. Roughly, an increase in λ by 0.1, corresponds to an increase in the implied volatility by about 0.08.

We then observe the effect of the mean of log(jump size) μ on the value of the



a. The effect of σ on the value of the performance fees.



b. The effect of λ on the value of the performance fees.

Figure 7.2: This figure shows the effect of the volatility σ and the mean jump arrival rate λ on the value of the performance fee. Observe that when b is far away from one, the value of the performance fees increases as σ or λ increases. When b is close to one, the value of the performance fees decreases as σ increases; however, the value of fees still increases as λ increases.

performance fees. The test results are shown in Figure 7.3. Observe that under typical parameters, the value of the performance fees decreases as μ increases. In order to analyze the effect of μ on the value of the performance fees, we rewrite the equation (2.13) as following

$$V_\tau = \frac{1}{2}\sigma^2 S^2 V_{SS} - (r+\lambda)V + V_S S(r - m_{total} - \kappa\lambda) + m_c S + \lambda \int_0^{+\infty} g(J)V(JS,t)dJ, \tag{7.1}$$

where $g(J) = \frac{\exp(-\frac{(\log(J)-\mu)^2}{2\gamma^2})}{\sqrt{2\pi}\gamma J}$, so $\kappa = E^Q[J-1] = \exp(\mu + \frac{\gamma^2}{2}) - 1$. From equation (7.1), we can see that when μ decreases, S has a higher probability to decrease as a jump occurs, which results in a higher probability of knock-out. A decrease in μ also results in a decrease in κ, hence a higher drift rate $(r - m_{total} - \kappa\lambda)$. So they have opposite effects on the value of the performance fees. From our test results in Figure 7.3, the effect of the increase in the drift rate dominates the effect of the higher probability of knock-out.

The effect of the standard deviation of log(jump size) γ on the value of the performance fees is shown in Figure 7.4. Observe that under typical parameters, the plot of the value and performance fees vs. γ has a smile shape, though the magnitude of the effect is much smaller than the other jump parameters λ and μ.

With jump			No jump		Difference	
λ	σ	Value	σ	Value	$\Delta\lambda$	$\Delta\sigma$
0.1	0.15	6.63372	0.230403	6.63373	0.1	0.080403
0.2	0.15	8.79043	0.314975	8.79044	0.2	0.164975
0.1	0.25	8.93663	0.321137	8.93665	0.1	0.071137
0.2	0.25	10.6359	0.400142	10.6359	0.2	0.150142
0.1	0.35	11.0549	0.42252	11.0549	0.1	0.07252
0.2	0.35	12.4224	0.50891	12.4224	0.2	0.15891

Table 7.1: This table shows the quantitative relationship between the implied volatility σ and the mean jump arrival rate λ. Observe that the values under the column "With jump" (the third column) are almost equal to the corresponding values under the column "No jump" (the fifth column). From the columns $\Delta\lambda$ and $\Delta\sigma$, we can see that an increase of 0.1 in λ has a similar effect to an increase of 0.08 in σ.

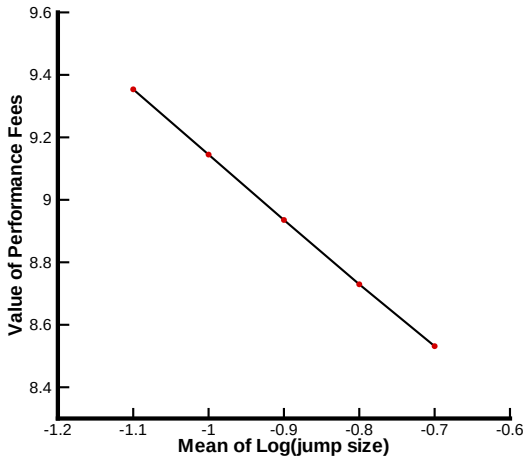


Figure 7.3: The effect of the mean of log(jump size) μ on the value of the performance fees.

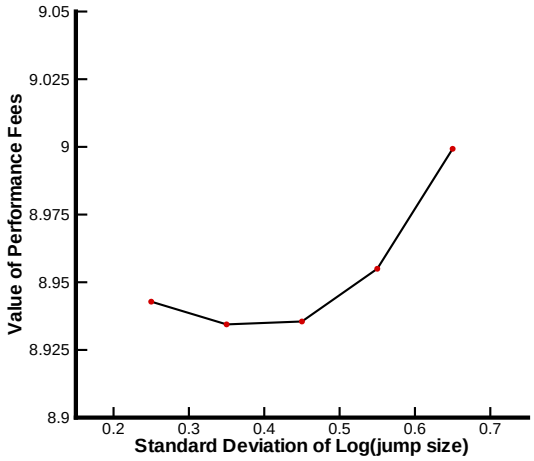


Figure 7.4: The effect of the standard deviation of log(jump size) γ on the value of the performance fees.

7.3 The Effect of Contractual Parameters

In this section, we will use the model to evaluate the proportion of the present value of the performance fees in the value of assets under the typical market and contractual parameters. Suppose that $\sigma = 0.25$, $\lambda = 0.1$, $p = 0.2$, $h = 0.05$ and the initial value of the asset is 100, then the present value of fees depends on the contract lifetime T and the endogenous choice of withdrawal level b. The test results are shown in Figure 7.5.

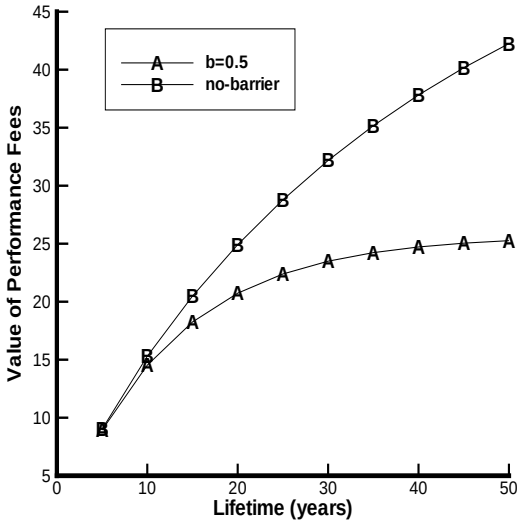


Figure 7.5: The effect of the contract lifetime T and the endogenous choice b on the value of the performance fees. We can observe that when T is small, the influence of the liquidation level is very small. When the lifetime is extended, the liquidation level has a large influence on the value of the performance fees.

From Figure 7.5, we can see that generally, the present value of the performance fees is a large fraction of the value of the assets under the typical parameters (40% no-barrier; 25% $b = 0.5$) when the lifetime is long. As the contract lifetime is extended, the liquidation level b plays a more important role. This is intuitively reasonable, since as the contract lifetime increases, the possibility of hitting the liquidation barrier also increases, and so the present value of the performance fees is more sensitive to the withdrawal behavior.

We then observe the effects of the performance rate p and hurdle rate h on the value of the performance fees. The test results are shown in Figure 7.6. From Figure 7.6, we can observe that when the performance rate increases, the value of the performance fees also increases and when the hurdle rate increases, the value of the performance fees decreases. Comparing these two parameters, we can see that the performance rate has a larger effect than the hurdle rate on the value of the performance fee.

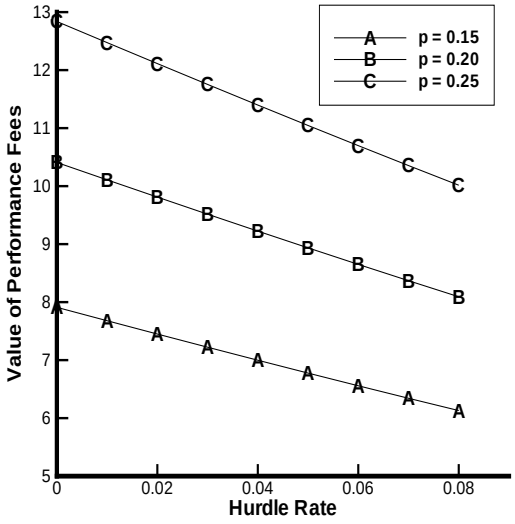


Figure 7.6: The effect of the performance rate p and hurdle rate h on the value of the performance fees.

7.4 Comparison Between Performance Fee and Fixed Rate Fee

At present, the option-like performance fee structure is widely used in hedge funds, while the fixed rate fee structure is widely used in mutual funds. In this section, we will compare these two structures quantitatively. We still assume that the investors withdraw their money if the asset price drops to half of the high-water

mark ($b = 0.5$). Consider that at some discrete times (once a year), instead of the performance fee, the manager charges a fixed rate fee $S \times f$, where S is the current price of asset, f is the fixed proportion. We wish to calibrate f to the present value of the performance fees.

Since we assume that investors withdraw their money if the asset price hits half of the high-water mark, we should keep the high-water mark H as an extra state variable. However, we do not need S_{old} any more. Denote the present value of the fixed rate fees to be V_f. Obviously, V_f also satisfies the equation (2.13). The only difference is the payoff and jump condition. Instead of the payoff equation (2.16), the payoff for fixed rate fees is

$$P(t_i) = S(t_i^-) \times f. \tag{7.2}$$

The jump condition is

$$V_f(S, H, t_i^-) = V_f(S^+, H^+, t_i^+) + P_i, \tag{7.3}$$

where

$$\begin{aligned} P_i &= S \times f, \\ S^+ &= S - P_i, \\ H^+ &= \max(H, S^+). \end{aligned} \tag{7.4}$$

Obviously, if f is known, we can use a similar numerical method to obtain V_f. Conversely, with the value of incentive performance fee V, we can use an iterative method to compute f such that $V_f = V$. Following this idea, we can evaluate the performance fee cash flow using the fixed rate cash flow. The result is shown in Table 7.2.

From Table 7.2, we see that if $\sigma = 0.15$, $\lambda = 0.1$, a performance fee rate of 10% is roughly equal in present value terms to a fixed rate fee of 0.75%. If we increase the volatility σ from 0.15 to 0.25, then a ten percent performance rate corresponds to a fixed rate of about 1.07 percent. The increase in λ from 0.1 to 0.2 also has a similar effect. This indicates that managers with an incentive fee structure have an obvious interest in raising the risk of underlying asset, e.g. through increasing σ or λ.

$Performance$ $rate$	$Present\ value\ of$ $performance\ fee$ $cash\ flow$	$Fixed$ $rate$	$Present\ value\ of$ $fixed\ rate$ $fee\ cash\ flow$
$\sigma = 0.15$, $\lambda = 0.1$			
0.1	3.3688	0.0075819	3.3689
0.2	6.63337	0.0151587	6.63336
0.4	12.8600	0.030303	12.8600
$\sigma = 0.25$, $\lambda = 0.1$			
0.1	4.56850	0.0107505	4.56850
0.2	8.93551	0.0215526	8.93551
0.4	17.0940	0.0433871	17.0940
$\sigma = 0.15$, $\lambda = 0.2$			
0.1	4.4890	0.010798	4.4890
0.2	8.7900	0.0216015	8.7900
0.4	16.8530	0.0432399	16.8530

Table 7.2: Comparison between the performance fee and the fixed rate fee. We can observe that the value of the performance fees (the second column) is almost equal to the corresponding value of fixed rate fee (the fourth column). Then observe the first column and the third column. In this case, if $\sigma = 0.15$, $\lambda = 0.1$, then a ten percent performance rate corresponds to about 0.75 percent for the fixed rate. If we increase the volatility σ from 0.15 to 0.25, then a ten percent performance rate corresponds to about 1.07 percent for the fixed rate. The increase in λ from 0.1 to 0.2 also has a similar effect.

Chapter 8

Parallel Computation

In this chapter, we will use two techniques for implementing parallel processing (μC++ and OpenMP) to speed up the hedge fund fee computation. In the test process, we use an existing option pricing library. The existing pricing library consists of about $50K$ lines of C++, developed by a number of professors and students over several years [27]. This primary design objective of this software was ease of extensibility for pricing new contracts. This objective was met through extensive use of inheritance and containers. The use of the μC++ language and OpenMP on a multiprocessor computer was not in the original design criteria. Hence, this is an interesting case study to demonstrate and compare the ease of use of the μC++ language and OpenMP for path dependent option pricing problems.

8.1 Numerical Method

In the existing option pricing library [27], the valuation of path dependent options reduces to a set of one-dimensional Partial Integral Differential Equations(PIDEs) embedded in a two or three-dimensional space [31]. In the computing process, the two or three-dimensional space is discretized. The greater the accuracy desired, the finer the two or three-dimensional grid. This means a problem in a three-dimensional space can have k^2 one-dimensional PIDE problems, where k is of the order of $50 - 100$.

The one-dimensional PIDEs are independent except at discrete observation times, which are called events. Between two adjacent event times, the one-dimensional

PIDEs advance in time to the next event. Upon reaching an event, these one-dimensional PIDEs exchange information, usually by some type of interpolation methods. In order to facilitate the discussion of the parallelization of the library, some of the important classes and functions in this library are described.

In the following, the class which solves a one-dimensional PIDE is called *OneD*. Its basic high-level operations are

- Advance-time: Each object of class *OneD* advances the solution from the previous event time to the next event time.
- Interpolation: At an event time, each object of class *OneD* may exchange information with other objects of the class *OneD*, usually by means of an interpolation operation. In the existing option pricing library, the interpolation operation is carried out by the interpolation function, which is a member routine of the *Event* class. The interpolation function is triggered at each event time.

Between any two adjacent event times, the advance-time operations are completely independent. The only difficulty is that all operations must share intermediate results at each event time.

The interpolation function consists of three parts. First, data is read from all the *OneD* objects. Then an interpolation operation is performed, and finally the new interpolated values are written back to the *OneD* objects. Note that all reading and interpolation operations are independent, as are all writing operations. However, reading and interpolating must finish before the next writing starts.

Clearly there are many operations which can be executed concurrently in a path dependent option pricing problem, which suggests the use of a multiprocessor computer to accelerate the computation.

In the following discussion, the no-arbitrage valuation of hedge fund fees is used as an example of a path dependent option to carry out testing. The technique discussed in this paper can be easily extended to other path dependent option pricing problems embedded in a higher or lower dimensional space, as discussed in [31, 32, 17, 13]. As a test example, we use the data given in Section 6.1 to do the parallel computation test (no similarity reduction used).

8.2 Algorithm for the μC++ Language

The μC++ language is an extension of the C++ programming language [26]. The extensions introduce new kinds of objects that augment the existing set of control flow facilities and provide for lightweight concurrency on a uniprocessor and parallel execution on a multiprocessor running the Unix or Linux operating system.

In μC++, a task and a monitor with condition variables and wait/signal statements are used to implement concurrency and synchronization. In μC++, a task is an object created from a task type in which a new thread of control is started in its main member routine called the task main. After a task is created, its thread starts to run automatically.

A monitor is an abstract data type with an implicit mutual-exclusion property, similar to a critical region, i.e., only one task at a time can be executing a monitor operation on its shared data. Tasks and monitors also have all the properties of classes.

A condition variable has to be owned by a monitor, which ensures that at most one task can operate with this condition variable at any time. The condition variable contains a queue on which tasks can be blocked and reactivated in FIFO order. When a task executes a wait statement of the condition variable, the active task is blocked and inserted in the queue of the condition variable. The blocked task is reactivated when another (active) task executes a signal statement of this condition variable (refer to [26] for more details concerning condition variables and wait/signal statements).

As discussed above, the hedge fund fees pricing problem includes three types of operations, the advance-time operation, which is carried out by the advance-time function in the *OneD* object, the interpolation operation, which is carried out by the interpolation function in the *Event* object and the control operation, which is carried out by the main function which initializes variables, releases memory, and so on. Tasks are used to carry out these three operations.

In μC++, each application has a main task named *uMain*. That is similar to the main routine in the C++. It is natural to let this task carry out control operations, such as creating necessary monitors and tasks.

For the advance-time and interpolation operations, it is not obvious how to carry out these operations with tasks. In the current architecture of the pricing software, these operations are naturally encapsulated in different classes. This design choice separates contractual features (which are specified in the interpolation function) from the stochastic process models (which are specified in the advance-time function). In general, the task design for the advance-time and interpolation functions should achieve the following criteria, as much as possible:

- Parallelize the necessary operations.
- Keep the functions related to the stochastic process models unchanged.
- Keep the functions related to contractual features unchanged.
- Design the tasks so as to allow easy extension to other path dependent option pricing problems.

8.2.1 Advance-time Function

As discussed earlier, the pricing problem has a set of one dimensional PIDEs embedded in a two- or three-dimensional space. From a software point of view, this is a set of container classes:

- $TwoD$ - contains a collection of $OneD$ objects.
- $ThreeD$ - contains a collection of $TwoD$ objects.

In this work, there is only one $ThreeD$ object, which contains many $TwoD$ objects. Each of these $TwoD$ objects contains many $OneD$ objects.

The advance-time function is a member of the $ThreeD$ class, which calls the advance-time functions of class $TwoD$ and class $OneD$. The actual computation is performed in the objects of the $OneD$ class.

There are at least three possibilities for designing the task for the advance-time function:

- Declare all *OneD* objects as tasks, and have them perform their advance-time functions in parallel.
- Declare all *TwoD* objects as tasks, and have them perform their advance-time functions in parallel.
- Define a pool of worker tasks to execute the advance-time function in the *OneD* objects.

The first option is to convert the *OneD* class into a task type, giving it its own thread of execution. This alternative is appealing because it reflects the parallelism inherent in the problem: an individual *OneD* task can perform the independent parts of its computation using its own thread, and synchronize with other tasks at well-defined points in order to communicate intermediate results. However, this design does not correspond well to the structure of the program as written. Much of the calculation logic falls outside the *OneD* class but makes use of data stored within the *OneD* object. Since multiple threads must access each object, they must be properly synchronized. However, the implicit synchronization provided by μC++ protects each individual operation on the *OneD* object and hence is too expensive, while disabling the implicit synchronization leads to a system whose correctness is difficult to verify. Without a significant restructuring of the system, this design is not an effective use of μC++.

Furthermore, even if the system were restructured to support this design, there is an additional limitation to this approach. A task requires more resources than a class, since it has an implicit execution context which requires memory, and a thread which adds scheduling and synchronization overhead. For this problem, a typical level of discretization results in more than 30000 *OneD* tasks. Preliminary experiments with this design show that this number of tasks has a significant effect on the scalability of the system.

Declaring all *TwoD* objects as tasks decreases the number of tasks running in parallel by a significant amount. For example, for the same level of discretization which gives rise to more than 30000 *OneD* objects, there are only about 180 *TwoD* objects. However, this method does not work for a pure two-dimensional option pricing problem, since in this case there is only one *TwoD* object. The other shortcoming of declaring *OneD* or *TwoD* objects as tasks is that both *OneD* and *TwoD* objects are kernel parts of the existing library, which are inherited by other

objects. However, in μC++, a task cannot be inherited by an ordinary class. It is unrealistic to convert all the derived classes to be task types. Because of these fatal shortcomings, these two designs are ruled out in future discussions.

The third option is to leave the *OneD* and *TwoD* objects as they are, and use a separate pool of worker tasks to execute the advance-time routine in the *OneD* objects. Consequently a new task which is called *AdvanceTask* is defined to carry out the advance-time function in the *OneD* objects. In order to decrease the overhead associated with the number of tasks and make use of the multiprocessor as much as possible, the number of tasks is set to be equal to the number of processors in use. However, this design method leads to task allocation problems.

Suppose that the number of tasks and the number of *OneD* objects are different. There are at least two possibilities for allocating *OneD* objects to tasks, dynamic allocation or static allocation. Dynamic allocation means that once a task is free and there is an unexecuted advance-time function in a *OneD* object, then the *OneD* object will be allocated to this free task and its advance-time function will be executed. Static allocation, on the other hand, means that all *OneD* objects are pre-allocated to different tasks. Since all tasks have to synchronize at an event time, static allocation will lead to the situation where the total running time is equal to the time spent by the most overloaded task. Dynamic allocation can avoid this problem. However the dynamic allocation process itself requires some overhead. There is a tradeoff in the choice of the allocation method. So the allocation criteria depends on whether the work can be distributed evenly. If the work can be distributed evenly (i.e. all *OneD* objects require the same work), then the obvious and optimal solution would be to statically allocate an equal number of *OneD* objects to each task. However, since the total work to be done for any time step is unequally distributed over the *OneD* objects (that is, some *OneD* objects require more work than others), it is worth the overhead of dynamic allocation in order to increase parallelism. Consequently, the dynamic allocation is used in the *AdvanceTask* tasks allocation and the monitor data structure is used to carry out the dynamic allocation.

8.2.2 Interpolation Function

The interpolation function mainly reads and writes data from the *OneD* objects. In the existing library, the interpolation function is a type of event. After all *OneD* objects reach an event time, the main routine calls the do_event function, which is a

virtual function of the *Event* class. The do_event function is redefined in the derived class of the *Event* class. For different events, depending on contract details, the redefined do_event function calls corresponding functions to deal with events. At present, there are many types of events that can be used directly by a user. These events enable the user to add new contract features easily. However, the present code for these existing events cannot be executed in parallel without modifying the software. Most of these events can be divided into three independent parts, reading, interpolating and writing. All of the reading, interpolating and writing operations can be done in parallel. However, all reading and interpolating must be completed before any writing can begin. The general strategy used to alter the events to expose concurrency is as follows:

- Define the event as a task.
- Divide the event into two parts, reading and writing (interpolation is often combined with reading).
- Use a synchronization technique to guarantee the reading completes before the next writing starts.

When the main routine calls the do_event function, instead of the original event function, the do_event calls a member routine in the newly defined tasks to activate the tasks to read data from the *OneD* objects. This enables the event to be executed in parallel. For this problem, the $InterpTask$ task is defined to implement the interpolation function, and the interpolation function is divided into a reading function and a writing function, which are member routines of the $InterpTask$ task. Unlike the advance-time functions in the *OneD* objects, all reading functions require almost the same work, as do all writing functions. So in this case, it is not worth paying the price for the overhead of dynamic allocation. Consequently a static method is used to allocate the reading and writing works to the $InterpTask$ tasks.

This requires defining a new task for each contract type. Since each new contract type requires a new class derived from the *Event* class in any case, this is not too onerous. However, for existing contract types, a new task must also be defined to achieve parallel execution, which violates one of our design criteria. Fortunately this set of alterations is fairly straightforward.

8.2.3 Tasks

The above implementation results in three type of tasks:

- *AdvanceTask* task - This task is a pool of worker tasks and is dynamically allocated to execute the advance-time functions in the *OneD* objects.
- *InterpTask* task - This task is mainly composed of two parts: a reading function and a writing function. The reading function reads data from the *OneD* objects, then uses interpolation to obtain new data which is written back to the *OneD* objects by the writing function. All the *InterpTask* tasks are statically allocated to read and write the data in the *OneD* objects.
- *uMain* task - This task initializes, deletes all *AdvanceTask* tasks and *InterpTask* tasks and administrates the synchronizations between tasks.

8.2.4 Synchronization Issues

To guarantee the correctness of the concurrent computation, three synchronization conditions must be guaranteed:

- The *InterpTask* tasks can only start after all *AdvanceTask* tasks reach an event time.
- All *AdvanceTask* tasks can only restart after all *InterpTask* tasks complete their writing functions.
- At each event time, all writing functions can only start after all reading functions finish.

In μC++, there are many ways to implement these synchronizations. For this problem, a monitor data structure with condition variables and wait/signal statements is used to implement these synchronizations.

8.3 Algorithm for OpenMP

OpenMP [43] is an Application Program Interface (API) that is primarily a directive-based method to invoke parallel computations on many shared-memory multiprocessor UNIX- and NT-based computers. OpenMP is comprised of three complementary components:

- A set of compiler directives used by the programmer to communicate with the compiler about parallelism.
- A runtime library which enables the setting and querying of parallel parameters such as the number of participating threads and the thread number.
- A limited number of environment variables that can be used to define runtime system parallel parameters such as the number of threads.

From the operating system point of view, the OpenMP functionality is based on the use of threads. In this report, the number of threads is set to be equal to the number of processors in use. An OpenMP application begins with a single thread, the master thread. As the program executes, the application may encounter parallel regions in which the master thread creates thread teams (which include the master thread). At the end of a parallel region, the thread teams are parked and the master thread continues execution. In a parallel region, there can be nested parallel regions where each thread of the original parallel region becomes the master of its own thread team. Nested parallelism can continue to further nest other parallel regions.

In OpenMP, from programmer's point of view, a loop is the basic unit of allocation of processors, so the following discussion is based on the loops in the source code. As discussed above, the pricing process mainly includes the advance-time operation and interpolation operation. The advance-time operation is composed of the loops that call advance-time functions in *OneD* objects [27]. The interpolation operation is composed of reading and writing loops. In order to parallelize the pricing process using OpenMP, there are three problems to be solved: parallelization, synchronization and allocation of threads.

8.3.1 Parallelization

In OpenMP, the primary means of parallelization is through the use of directives inserted in the source code. One of the most fundamental and most powerful directives is loop-level parallelism. In C/C++, if the clause *#pragma omp parallel for* is inserted before a *FOR* loop, then the *FOR* loop is to be executed in parallel. Note that there is a limit in this parallelization mode: the iterations in the loop must be independent. Fortunately, the iterations in advance-time loops are all independent, as are the iterations of interpolation operations (reading and writing loops). Consequently, it is very easy to parallelize the advance-time and interpolation operations using OpenMP, we only need to insert the *#pragma omp parallel for* clause before the loops of the advance-time and interpolation operations (including the reading and writing loops).

8.3.2 Synchronization

Similar to the μC++ case, in order to guarantee the correctness of the concurrent computation, the three synchronization conditions described in Section 8.2.4 must be satisfied.

All these three synchronization conditions require that after each thread finishes a loop (advance-time, reading or writing), it has to wait until all other threads have reached this point, and the master thread then continues. In OpenMP, there is an implied barrier at the end of a loop with a for directive, so no special code is necessary for the synchronizations.

8.3.3 Allocation of Threads

As discussed in Section 8.2.1 and 8.2.2, it is best to assign threads to the advance-time operation dynamically, and to the interpolation operation statically.

In OpenMP, it is easy to apply the dynamic or static assignment of threads by adding an option to the parallelization directives, for example,
#pragma omp parallel for schedule(*dynamic*).

8.4 Running Time Analysis

First, consider the case of using one processor to process serially all operations. The CPU time should satisfy

$$\begin{aligned} T_{total}^{uni} &= T_{init} + T_{advance} + T_{read} + T_{write} \\ &= T_{init} + T_{advance} + T_{interp}, \end{aligned} \tag{8.1}$$

where

T_{total}^{uni}	is the total CPU time on a uniprocessor computer,
T_{init}	is the CPU time for the initializing operation,
$T_{advance}$	is the total time spent on executing the advance-time functions,
T_{read}	is the total time spent on the reading function at an event time,
T_{write}	is the total time spent on the writing function at an event time,
T_{interp}	is the total execution time for the interpolation operations, which is equal to the sum of T_{read} and T_{write}.

Now consider the case of using a multiprocessor to process all operations in parallel. Suppose the number of processors is c. All advance-time operations can run independently during the interval between two adjacent event times. At each event time, all reading operations are independent, as well as the writing operations. Ideally, the time spent on the advance-time functions, the reading and the writing functions can be decreased by a factor of c. Here the execution time of each advance-time function for each *OneD* object is assumed to be identical. Similarly, suppose that the reading and writing functions require the same CPU time for every *OneD* object. However as the number of tasks and processors increase, the time spent on overhead also increases. T_{extra} is used to represent this extra overhead, so that

$$T_{total}^{multi} = T_{init} + \frac{T_{advance} + T_{interp}}{c} + T_{extra}, \tag{8.2}$$

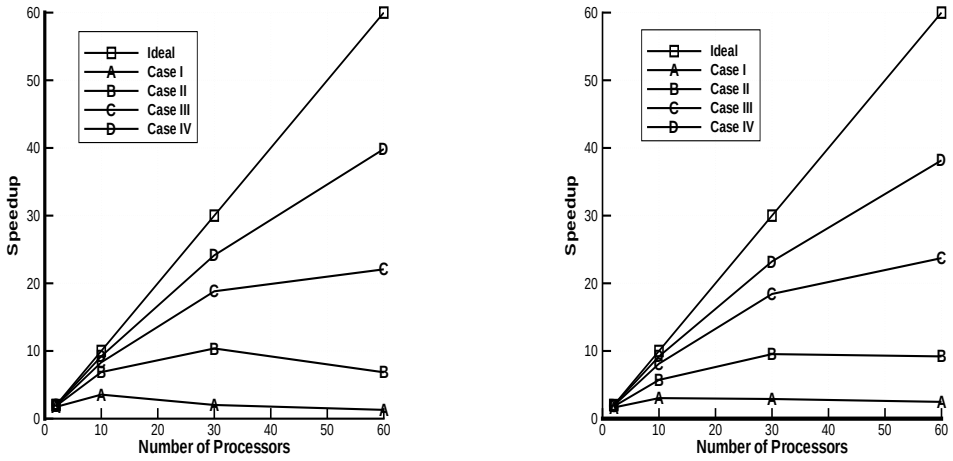
where

T_{total}^{multi}	is the total CPU time on a multiprocessor computer,
$T_{advance}$	is the total execution time for the advance-time functions on a uniprocessor computer,
T_{interp}	is the execution time for the interpolation operation (i.e. reading and writing functions) on a uniprocessor,
T_{extra}	is the extra time spent on multiprocessor and task overhead,
c	is the number of processors.

8.5 Test Results

In this section, μC++ and OpenMP are used to test the parallelization of the hedge fund fees pricing problem using different numbers of processors. All tests are carried out on an SGI machine with SUSE Linux (kernel version: 2.6.5-7.244-sn2), 64 processors and 128G memory. Four different cases are tested. Each case is denoted by the combination of the number of $OneD$ objects, the number of nodes in each $OneD$ object, and the time step. For example, in $Case\ I$, there are 552 $OneD$ objects and each of them has 35 nodes and the time step is 1/8 years, which is denoted by $[552, 35, 1/8]$. $Case\ II$ is the combination $[2115, 69, 1/16]$, $Case\ III$ is the combination $[8277, 137, 1/32]$, and $Case\ IV$ is the combination $[32745, 273, 1/64]$.

First, the original sequential version was run on the four cases. Next, the two concurrent versions were run on each of the four cases, with varying number of processors. In the multiprocessor tests, the speedup effects of the number of processors using μC++ and OpenMP are determined. These are shown in Figure 8.1, where $Speedup = T^{uni}_{total}/T^{multi}_{total}$ (T^{uni}_{total} and T^{multi}_{total} are total CPU times using uniprocessor and multiprocessor respectively). The straight line is the ideal situation using multi-processors. Figure 8.1(a) presents the test results using the μC++ language. Figure 8.1(b) shows the test results using OpenMP. In each figure, as the problem size increases, fixing the number of processors, the speedup improves. Comparing Figure 8.1(a) with (b), μC++ has a similar speedup compared to OpenMP.



a. (the μC++ case)

b. (the OpenMP case)

Figure 8.1: This figure shows the speedup using multi-processors compared with a uni-processor. The straight line is the ideal situation using multi-processors. Figure 8.1(a) shows test results using the μC++ language, Figure 8.1(b) shows test results using OpenMP.

8.6 Evaluation

From the performance test of the hedge fund performance fee pricing problem using the μC++ language and OpenMP, the following conclusions can be obtained:

- Both the μC++ language and OpenMP on a multiprocessor can greatly improve the performance of pricing path dependent contingent claims.
- As the number of processors increases, the proportion of overhead in the total running time also increases.

From the performance point of view, μC++ and OpenMP are similar. From the implementation point of view, OpenMP is superior. The reasons are the following:

- In OpenMP, no new concepts are needed to carry out the parallelization.
- In OpenMP, the allocation of the iterations to the threads are transparent. In μC++, programmers have to allocate actual computation work to tasks by hand.
- In OpenMP, the synchronization is implicit.

In most programs, the execution of loops costs most of the CPU times. In OpenMP, an iteration is the basic unit of allocation to processors, while in μC++, a task is the basic unit of allocation to processors. When modifying an existing serial program, where loops account for most of the CPU times, OpenMP would appear to have some advantages compared to μC++, since the compiler of OpenMP can allocate the iterations to the processors directly.

However, this convenience comes at the cost of flexibility. OpenMP does not work if the iterations are not independent. In this case, we can use μC++ to realize parallelism. From this point of view, μC++ is more flexible and powerful.

Chapter 9

Conclusion

In this chapter, we present a summary of the major contributions of this thesis, along with a discussion for possible further research.

9.1 Main Results

Hedge funds are an interesting investment class with an unusual form of manager compensation. The high-water mark provision creates a distinct option-like feature to the contract. In this thesis, we provide a mathematical model to quantitatively analyze the cost of a hedge fund manager contract and examine its implication. We find that if the lifetime of the contract is long enough, depending on the withdrawal strategy, the performance fee effectively "costs" the investors almost 25 to 40 percent of the portfolio under the typical market parameters. For short term contracts, the cost is more on the order of $5\% - 10\%$. By comparing the incentive plan with a fixed rate contract and quantifying the relationship between these two compensation plans, we show that the manager with the incentive fee has an interest in increasing risk, provided other non-modelled considerations are not overriding.

In this thesis, we analyze and summarize some effective interpolation methods and grid construction techniques which can improve the efficiency of numerical method. We also provide some basic guidelines for interpolation and grid arrangement when we solve a path-dependent pricing problem using a numerical method.

We then verify the accuracy of our numerical method using the MC method and

provide a fast and accurate MC algorithm to price the performance fee with jump diffusion, and a withdrawal barrier.

Finally we also analyze and apply two parallel computation techniques (μC++ and OpenMP) to a existing option pricing library, and compare these two parallel techniques when solving a high dimensional path-dependent pricing problem.

9.2 Future Study

In this thesis, we assume that the volatility is a constant. As we know, this is not very realistic. In the future we may use stochastic volatility with jump diffusion to model the asset price.

As discussed above, when an investor puts money into a hedge fund, he actually provides a free option to the manager of the hedge fund. Although the manager can receive the option value for sure without any special skill, this incentive plan is still widely used in hedge funds. The only explanation is that the investor believes that the hedge fund manager can outperform the market (given the level of risk assumed). We can use an equilibrium model to evaluate the manager's skill from the investor's viewpoint and check that whether the "free option" is justified. Actually, some work has been done in [12] in this direction, but the authors suppose that the asset price follows geometric Brownian motion. We may add jump diffusion into the equilibrium model in the future.

Appendix A

Monte Carlo algorithm for generating a sample path under a jump diffusion process

Generating one sample path of jump diffusion in $\Delta t = t_{i+1} - t_i$

$T_{elapsed} \Leftarrow \Delta t^*$ // Δt^* is the interval between the next jump event time and t_i
if $T_{elapsed} > \Delta t$ then
  Generate standard normally distributed random number ϕ
  $S_{end} \Leftarrow S_{start} \times \exp[(r - m_{total} - \lambda\kappa - \frac{\sigma^2}{2})\Delta t + \sigma\phi\sqrt{\Delta t}]$
  if $S_{end} < barrier$ then
    return (-1.0) // knock out
  end if
  Compute the probability P of no barrier crossing
  during interval Δt according to (5.2)
  Generate uniformly distributed random number u in [0,1]
  if $u > P$ then
    return (-1.0) // knock out
  end if
else
  while $T_{elapsed} < \Delta t$ do
    Generate standard normally distributed random number ϕ
    $S_{end} \Leftarrow S_{start} \times \exp[(r - m_{total} - \lambda\kappa - \frac{\sigma^2}{2})\Delta t^* + \sigma\phi\sqrt{\Delta t^*}]$

continued on the next page

Generating one sample path of jump diffusion in $\Delta t = t_{i+1} - t_i$

continue from the previous page

```
    if S_end < barrier then
       return (−1.0) // knock out
    end if
    Compute the probability P of no barrier crossing
    during interval Δt* according to (5.2)
    Generate uniformly distributed random number u in [0,1]
    if u > P then
       return (−1.0) // knock out
    end if
    Generate lognormally distributed random number R, where R ~ LN(μ, γ)
    S_end ⇐ S_end × R
    if S_end < barrier then
       return (−1.0) // knock out
    end if
    Generate Δt* from the exponential distribution with mean 1/λ
    T_elapsed ⇐ T_elapsed + Δt*
  end while
  Δt_left ⇐ Δt − (T_elapsed − Δt*)
  Generate standard normally distributed random number φ
  S_end ⇐ S_end × exp[(ξ − λκ − σ²/2)Δt_left + σφ√Δt_left]
  if S_end < barrier then
     return (−1.0) // knock out
  end if
  Compute the probability P of no barrier crossing during interval Δt*
  according to (5.2)
  Generate uniform distributed random number u in [0,1]
  if u > P then
     return (−1.0) // knock out
  end if
end if
Δt* ⇐ T_elapsed − Δt // Δt* is now the interval between
the next jump event time and t_{i+1}
return S_end // S_end is the price at time t_{i+1}
```

Bibliography

[1] C. Ackermann, R. McEnally and D. Ravenscraft *The performance of hedge funds: risk, return, and incentives.* The Journal of Finance, Vol. 54 Issue 3 (June 1999) pp. 833-874.

[2] W. Fung and D.A. Hsieh *Performance characteristics of hedge funds and commodity funds: natural vs. spurious biases.* Journal of Financial and Quantitative Analysis, Vol. 35 (2000) pp. 291-307.

[3] W. Fung and D.A. Hsieh *A primer on hedge funds.* Journal of Empirical Finance, Vol. 6 (1999) pp. 309-331.

[4] W. Fung and D.A. Hsieh *Empirical characteristics of dynamic trading strategies: the case of hedge funds.* Review of Financial Studies, Vol. 10 (1997) pp. 275-302.

[5] T. Schneeweis and R. Spurgin *Mutlifactor analysis of hedge fund, managed futures, and mutual fund return and risk characteristics.* Journal of Alternative Investments, Vol. 1 (1998) pp. 1-24.

[6] S.J. Brown, W.N. Goetzmann and J. Park *Conditions for survival: changing risk and the performance of hedge fund managers and CTAs.* Working paper, International Center for Finance, Yale School of Management (1997).

[7] W.N. Goetzmann, J.E. Ingersoll Jr. and S.A. Ross *High water marks.* Working paper, Yale School of Management (1997).

[8] M. Grinblatt and S. Titman *Adverse risk incentives and the design of performance-based contracts.* Management Science, Vol. 35 (1997) pp. 807-822.

[9] J. Carpenter *The optimal investment policy for a fund manager compensated with an incentive fee.* Working paper, Stern School of Management, NYU (1997).

[10] J. Carpenter *Does option compensation increase managerial risk appetite?* Journal of Finance, Vol. 55 (2000) pp. 2311-2331.

[11] S. Basak, A. Pavlova, and A. Shapiro *Offsetting the incentives: risk shifting and benefits of benchmarking in money management.* Working paper, MIT (2003).

[12] W.N. Goetzmann, J.E. Ingersoll Jr. and S.A. Ross *High-water marks and hedge fund management contracts.* The Journal of Finance, Vol. 58 Issue 4 (August 2003) pp. 1685-1717.

[13] M.A.H. Dempster and J.P. Hutton *Fast numerical valuation of American, exotic and complex options.* Applied Mathematical Finance, Vol. 4 (1997) pp. 1-20.

[14] W. Fung and D.A. Hsieh *The risk in hedge fund strategies: theory and evidence from trend followers.* Review of Financial Studies, Vol. 41 (2001) pp. 313-341.

[15] C. He, J.S. Kennedy, T. Coleman, P.A. Forsyth, Y. Li and K.R. Vetzal *Calibration and hedging under jump diffusion.* Submitted to Review of Derivatives Research (2006).

[16] P. Wilmott *Paul Wilmott on quantitative finance.* John Wiley and Sons Ltd, Chichester, (2000).

[17] P. Wilmott *Cliquet options and volatility models.* Wilmott Magazine, December (2002) pp. 78-83.

[18] H.A. Windcliff, P.A. Forsyth, and K.R. Vetzal *Numerical methods and volatility models for valuing cliquet options* Working paper, School of Computer Science, University of Waterloo, (2006).

[19] 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, Vol. 25 (2005) pp. 87-112.

[20] H.A. Windcliff, P.A. Forsyth and K.R. Vetzal *Analysis of the stability of the linear boundary condition for the Black-Scholes equation.* The Journal of Computational Finance, Vol. 8:1 (Fall, 2004) pp. 65-92.

[21] P. Glasserman *Monte Carlo methods in financial engineering.* Springer, (2003).

[22] P.A. Forsyth, K.R. Vetzal, and R. Zvan *Convergence of lattice and PDE methods for valuing path dependent options with interpolation.* Review of Derivatives Research, Vol. 5 (2002) pp. 273-314.

[23] P. Wilmott, J. Dewynne and S. Howison *Option pricing: mathematical models and computation.* Oxford Financial Press, January, (2000).

[24] H.A. Windcliff *Computational methods for valuing path-dependent derivatives.* PhD thesis, School of Computer Science, University of Waterloo (2003).

[25] Y. d'Halluin *Numerical methods for real options in telecommunications.* PhD thesis, School of Computer Science, University of Waterloo (2004).

[26] P.A. Buhr and R.A. Stroobosscher *$\mu C++$ annotated reference manual.* School of Computer Science, University of Waterloo, (2005).

[27] P.A. Forsyth, H.A. Windcliff and K.R. Vetzal *PathDepPricer a user's overview.* School of Computer Science, University of Waterloo, (2005).

[28] G. de Brouwer *Hedge funds in emerging markets.* Cambridge University Press, (2001).

[29] S.J. Brown, W.N. Goetzmann and R.G. Ibbotson *Offshore hedge funds, survival and performance:1989-1995.* Journal of Business, Vol. 72, No. 1 (Jan.,1999) pp. 91-118.

[30] R.C. Merton *Option pricing when underlying stock returns are discontinuous.* Journal of Financial Economics, Vol. 3 (1976) pp. 125-144.

[31] R. Zvan, P.A. Forsyth and K.R. Vetzal *Robust numerical methods for PDE models of Asian options.* The Journal of Computational Finance, Vol. 1 (Winter 1998) pp. 39-78.

[32] K.R. Vetzal and P.A. Forsyth *Discrete Parisian and delayed barrier options: a general numerical approach.* Advances in Futures and Options Research, Vol. 10 (1999) pp. 1-16.

[33] R. Zvan, K.R. Vetzal and P.A. Forsyth *PDE methods for barrier options.* Journal of Economic Dynamics and Control, Vol. 24 (2000) pp. 1563-1590.

[34] I. Karatzas and S.E. Shreve *Brownian motion and stochastic calculus.* New York: Springer-Verlag, (1991).

[35] S. Metwally and A. Atiya *Using Brownian bridge for fast simulation of jump-diffusion processes and barrier options.* Journal of Derivatives, Vol. 10 (Fall 2002) pp. 43-54

[36] P. Boyle, M. Broadie and P. Glasserman *Monte Carlo methods for security pricing.* Journal of Economic Dynamics and Control, Vol. 21 (1997) pp. 1267-1321.

[37] D. Higham *An algorithmic introduction to numerical simulation of stochastic differential equations.* SIAM Review, Vol. 43 (2001) pp. 525-546.

[38] R. Rannacher *Finite element solutions of diffusion problems with irregular data.* Numerical Mathematics, Vol. 43 (1984) pp. 309-327.

[39] L. Walbein *A remark on parabolic smoothing and finite element methods.* SIAM Journal on Numerical Analysis, Vol. 17 (1980) pp. 33-38.

[40] R.G. Ibbotson and P. Chen *Sources of hedge fund returns: alphas, betas, and costs.* Working paper, International Center for Finance, Yale School of Management (2005).

[41] P.A. Forsyth and K.R. Vetzal *Computational finance.* CS870, course notes (2005).

[42] R.V. Hogg and A.T. Craig *Introduction to mathematical statistics.* Prentice Hall, (1995).

[43] D. Sondak, K. Tseng and J.T. von Hoffman *Introduction to OpenMP.* Scientific Computing and Visualization, Boston University.