Interpolation and American options pricing

Shi-hau Liao
Department of finanace
National Taiwan University

Contents

List of Figures

Abstract

Pricing European and American options accurately and efficiently has been a main concern in many studies. Although the closed-form solution of the European option has already been derived by Fischer Black, Myron Scholes, and Robert Merton and efficient numerical approximation algorithms are available, there are numerical methods that price such options with a much smaller cost and within acceptable error bounds by use of some precomputation.

In the thesis, the method is proposed to build a look-up table for European and American option values by precomputation. Once this is done, the requested option value is then interpolated from the table via polynomial interpolation or cubic spline. Though it takes time to build up the table, since the calculation is done off-line and once and for all, the cost is fixed and can be amortized. More importantly, the interpolated option value can be calculated very fast.

Chapter 1

Introduction

Introduction

TAIEX options and equity options have already been issued in recent years, and received great attention from the investors, mutual funds, financial institutions, and market makers. If the investors, mutual funds, and financial institutions cannot get the correct prices of the options on time, they may not know the costs of hedging, seize the arbitrage opportunities, or rebalance their portfolios. For market makers, not being able to quote the accurate bid and ask price promptly means they may fail to get the orders and consequently lose a potentially lucrative deal. Therefore, it is very important for them to price the options fast and accurately.
The method in the thesis is proposed to build a look-up table for European and American option values by precomputation. Once this is done, the requested option value is then interpolated from the table via polynomial interpolation or cubic spline. Though it takes time to build up the table, since the calculation is done off-line and once and for all, the cost is fixed and can be amortized. More importantly, the interpolated option value can be calculated very fast.

Previous work

The method of building a look-up table of the American option prices and calculating our desired option prices by interpolation was first introduced by Broadie and Detemple in Recent advances in numerical methods for pricing derivative securities and was first implemented by Adriaan Joubert and L.C.G Rogers in Fast, Accurate and Inelegant Valuation of American Options, in which they use polynomial interpolation to maintain its accuracy.

Structures of the Thesis

The structure of this thesis is as followed.Section 2 focuses on the definition of the American and European options.Secton 3 introduces two interpolation skills: polynomial interpolation and cubic spline.Section 4 describes the way to build the look-up table.Section 5 displays the numerical results.

Chapter 2

Backrounds

2.1 Derivatives Basics

This section covers the definition and classification of the option and the method to price them.

2.1.1 Option Basics

An option is the right to buy or sell a specified underlying asset at a specified price within a specified period of time. Generally speaking, there are two basic types of option :*call options, put options*. A call option is the right to buy a security at a specified price (called the *exercise* or *strike price*) during a specified period of time, while a put option is the right to sell a security at a specified price during a specified period of time. The price which holder can buy or sell something is called the *exercise price* or the *strike price*. The date which the contract expires is known as the *expiration date*, *exercise date* or *maturity*.

The options can also be differentiated by the time period when they can be exercised. An *American option* can be exercised at any time up to the expiration date, whereas a *European option* can be exercised only at expiration. Thus, the price of an*American option* must be at least as much of that of a*European option* with the same exercise price and time to maturity because of its early exercise feature.

There are two positions an investor may take. One is a*short position*, the other is a*long position*. An investor taking a long position on an option means he buys the option, while an investor taking a short position on an option means he sells the option. Taking a long position on the call options means the investor expects the price of the underlying asset to rise, while taking a short position on the call option means the investor expects the underlying asset to decline.

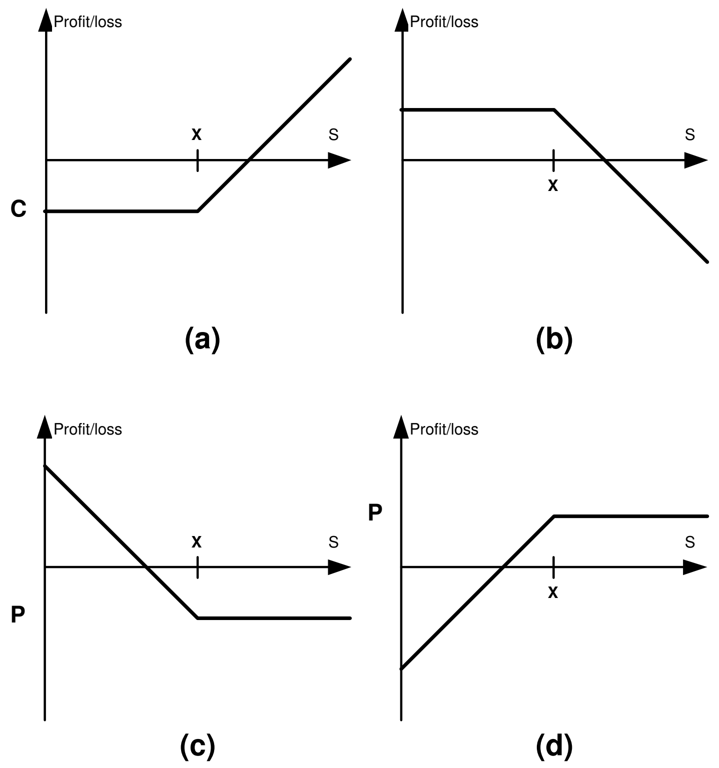

Figure 2.1: Profit/loss of Options. (a) Long a call. (b) Short a call. (c) Long a put. (d) Short a put.

2.1.2 Payoffs on Standard Options

Not obliging the option holder to exercise the right, an option will be exercised only when its profit is maximized. Therefore, a call option will be exercised when the price of the underlying asset is lower than its strike price; in contrast, a put option will be exercised when the price of the underlying asset is higher than its strike price. To define it more formally, we let the value of the underlying asset at maturity be S, the strike price be X, and the premium of option be O. Then the payoff for the long position at expiration is $\max(0, S-X)$ for European call options; and $\max(0, X-S)$ for European put options. So the profit for a long position in call options is

$$\max(0, S-X) - O$$

The profit for a long position in put options is

$$\max(0, X-S) - O$$

The profit for a short position in call options is

$$-(\max(0, S-X) - O) = \min(0, X-S) + O$$

while the profit for a short position in put options is

$$-(\max(0, X - S) - O) = \min(0, S - X) + O$$

Figure2.1 illustrates profit/loss graphically.

2.2 Pricing Methods

2.2.1 The Balck-Scholes Formula

In the early 1970s, Fischer Black and Myron Scholes made a major breakthrough in the pricing of non dividend-paying derivative by deriving a well-known differential equation. Solving the differential equation results in the closed form solution of European call and European put option on stock, which is one of the most significant tools for pricing options.

2.2.1.1 Assumptions

The assumptions used to derive the Black-Scholes differential equation are listed below:

1. The value of the underlying assets follows the log-normal distribution.
2. The rate of return on stock, μ, and the volatility of stock price, σ, are constant throughout the option's life.
3. The short selling of securities with full use of proceeds is permitted.
4. The are no transaction costs or taxes. All securities are perfectly divisible.
5. No dividends are paid during the life of the derivative security.
6. No arbitrage opportunity.
7. Security trading is continuous.
8. The risk-free rate of interest, r, is constant and the same during the life of the security.

2.2.1.2 The Black-Scholes Differential Equation

From assumptions 1 and 2 above, we know that

$$dS = \mu S dt + \sigma S dz$$

where dz follows $N(0, dt)$, S denotes the stock price and dt denotes a very short time interval. By the Ito's formula

$$df = (\frac{\partial f}{\partial S}\mu S + \frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2})dt + \frac{\partial f}{\partial S}\sigma S dz$$

where f is the price of a call option or other derivative contingent on S. We may use S and f to form a portfolio without dz, the random source of the underlying stochastic process, through a suitable choice of weights for each asset. Because the portfolio is riskless, it earns the risk-free rate during the short time interval, dt. After arranging the formula, the final equations emerges as

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf \tag{2.1}$$

where f is the price of a derivative security, S is the stock price, σ is the volatility of the stock price, and r is the continuously compounded risk-free rate.

2.2.1.3 The Closed Form Solution for The Black-Scholes Formula

The closed form solutions for the price of European calls and puts by solving (2.1) along with the boundary condition,

$$f = \max(S - X, 0) \text{ for the European call option}$$

$$f = \max(X - S, 0) \text{ for the American call option}$$

are

$$C = SN(d_1) - Xe^{-rT}N(d_2)$$

$$P = Xe^{-rT}N(-d_2) - SN(-d_1)$$

where

$$\begin{aligned} d_1 &= \frac{\ln(S/X) + (r + \sigma^2/2)T}{\sigma\sqrt{T}} \\ d_2 &= \frac{\ln(S/X) + (r - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T} \end{aligned}$$

The notations for the above equation are described as below:
C denotes the call price
P denotes the put price
$N(x)$ = Probability distribution function for standard normal distribution
σ^2 = Annualized variance of the continuously compounded return on stocks
r = Continuously compounded risk-free rate
T = The time to maturity

2.2.2 Binomial Option Pricing Model

First, we consider the single period binomial model, that is, the call option matures at time one. Let S denote the stock price at time zero, X be the strike price, u and d [1]denote the percentage that stock price move up and down at time one respectively,and C_u and C_d be the price of the call option when stock move up to Su and Sd at time one respectively.Thus

$$C_u = \max(0, Su - X)$$

$$C_d = \max(0, Sd - X)$$

Now we want to set up a portfolio with h shares of stock and B dollars in the riskless asset so that the payoff of the portfolio replicate that of the call, that is,

$$hSu + B = C_u$$

$$hSd + B = C_d$$

By solving the above equation we obtain

$$h = \frac{C_u - C_d}{Su - Sd}$$

$$B = \frac{uC_d - dC_u}{(u-d)R}$$

where R is the risk-free rate. Therefore,

$$hS + B = \frac{(\frac{R-d}{u-d})C_u + (\frac{u-R}{u-d})C_d}{R}$$

which can be rewritten as

$$hS + B = \frac{pC_u + (1-p)C_d}{R}$$

where

$$p \equiv \frac{R-d}{u-d}$$

Likewise, in the two period model, the value of the call option at time two shall be

$$C_{uu} = \max(0, Suu - X),\ C_{ud} = \max(0, Sud - X),\ C_{dd} = \max(0, Sdd - X)$$

Applying the same logic in the one period model, we may get the value of the call option at time one:

$$C_u = \frac{pC_{uu} + (1-p)C_{ud}}{R},\ C_d = \frac{pC_{ud} + (1-p)C_{dd}}{R}$$

[1] u and d reflect the volatility of the price of underlying asset, the most common choice of u and d are $e^{\sigma dt^{0.5}}$ and $e^{-\sigma dt^{0.5}}$.

Therefore,

$$C = hS + B = \frac{pC_u + (1-p)C_d}{R} = \frac{p^2C_{uu} + 2p(1-p)C_{ud} + (1-p)^2C_{dd}}{R^2}$$

$$= \frac{p^2 \max(0, Su^2 - X) + 2p(1-p)\max(0, Sud - X) + (1-p)^2 \max(0, Sd^2 - X)}{R^2}$$

To extend it more generally, we let $S(i,j)$ denote the price of underlying asset with i up and $i-j$ down moves from the root, that is,

$$S(i,j) = S_0 u^j d^{i-j}$$

and let $C(i,j)$ and $P(i,j)$ be the price of European call and European put option when the price of the underlying asset is $S(i,j) = S_0 u^j d^{i-j}$. Then, we may write down the induction formula of the call option:

$$C(i,j) = \frac{pC(i+1,j+1) + (1-p)C(i+1,j)}{R}$$

for $i = 0, 1, ..., n$ and $j = 0, 1, ..., i$, through which we may get the value of call option in the n period model:

$$C(0,0) = \frac{\sum_{j=0}^{n}\binom{n}{j}p^j(1-p)^{n-j}\max(0, Su^j d^{n-j} - X)}{R^n}$$

Similarly, the value of a European put is

$$P(0,0) = \frac{\sum_{j=0}^{n}\binom{n}{j}p^j(1-p)^{n-j}\max(0, X - Su^j d^{n-j})}{R^n}$$

For the American call and put options, the above equation should be modified because it can be exercised at any time before expiration. The induction formula of American call option turns out to be

$$C(i,j) = \max(\frac{pC(i+1,j+1) + (1-p)C(i+1,j)}{R}, S(i,j) - X)$$

and that of the American put option turn out to be

$$P(i,j) = \max(\frac{pP(i+1,j+1) + (1-p)P(i+1,j)}{R}, X - S(i,j))$$

Chapter 3

Polynomial and Cubic Spline Interpolation

3.1 Polynomial Interpolation

The idea of polynomial interpolation is simple. Suppose we are given $n+1$ points $P_0(x_0, y_0), P_1(x_1, y_1), ..., P_n(x_n, y_n)$, which we shall refer to as knots. No restrictions are imposed on the $y_k s$. But we do assume that $x_k s$, which we shall refer to as nodes, are distinct and in their natural order, that is, $x_0 < x_1 < \cdots < x_k < x_{k+1} < \cdots < x_n$. The objective is to find polynomials that interpolate one or more of these knots.
To define it more generally, we let P_i be the value at x of the unique polynomial of degree zero passing through the point (x_i, y_i) and let $P_{i,i+1}$ be the value at x of the unique polynomial of degree one passing through both (x_i, y_i) and (x_{i+1}, y_{i+1}). Similarly for higher-order polynomials, up to $P_{i,i+1,i+2,i+3....i+m}$. Let $p_{k,k+m}(x)$ denote the polynomial passing through $(m+1)$ knots,$P_k, P_{k+1}, ..., P_{k+m}$, that is, $p_{k,k+m}(x_i)$ imposes $m+1$ constraints $p_{k,k+m}(x_i) = y_i$ for $i = k, k+1, ..., k+m$. This suggests that $p_{k,k+m}$ has at most $m+1$ coefficients, that is, $p_{k,k+m}(x)$ is of degree at most m. For example, when $m = 0$, $p_{k,k}(x)$ is the zeroth-degree polynomial whose graph is horizontal line through the one knot $P_k(x_k, y_k)$, that is, $p_{k,k}(x)$ is a constant function. When $m = 1$, $p_{k,k+1}(x)$ is the first-degree polynomial whose graph is the unique straight line through $P_k(x_k, y_k)$ and $P_k(x_k, y_{k+1})$, that is, $p_{k,k+1}(x) = y_k + \frac{y_{k+1}-y_k}{x_{k+1}-x_k}(x - x_k)$.
We can further express the polynomial in the Lagrange's form.

$$p_{k,k+m}(x) = \frac{(x-x_k)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_{k+m})}{(x_i-x_k)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_{k+m})}y_i$$
$$+\frac{(x-x_k)\cdots(x-x_i)(x-x_{i+2})\cdots(x-x_{k+m})}{(x_{i+1}-x_k)\cdots(x_{i+1}-x_i)(x_{i+1}-x_{i+2})\cdots(x_{i+1}-x_{k+m})}y_{i+1}$$
$$+\cdots+\frac{(x-x_k)(x-x_{k+1})\cdots(x-x_{k+m-1})}{(x_{k+m}-x_k)(x_{k+m}-x_{k+1})\cdots(x_{k+m}-x_{k+m-1})}y_{k+m}$$

There are $m+1$ terms, each a polynomial of degree m and each constructed to be zero at all of the x_i $i = k, k+1, ..., k+m$ except one, at which it is constructed to be y_i $i = k, k+1, ..., k+m$.
To implement the Lagrange formula efficiently, we may use the following relationship between a "daughter" P and its two "parents,"

$$P_{i,(i+1)...(i+m)} = \frac{(x - x_{i+m})P_{i,(i+1)...(i+m-1)} + (x_i - x)P_{(i+1),(i+2)...(i+m)}}{x_i - x_{i+m}} \tag{3.1}$$

This recurrence works because the two parents already agree at points $x_{i+1}, ..., x_{i+m-1}$. The various Ps form a "tableau" with "ancestors" on the left leading to a single "descendant" at the extreme right. For example, with $m = 4, i = 1$

$$\left\{\begin{array}{lllll}
x_1: & y_1 = P_1 & & & \\
 & & P_{1,2} & & \\
x_2: & y_2 = P_2 & & P_{1,2,3} & \\
 & & P_{2,3} & & P_{1,2,3,4} \\
x_3: & y_3 = P_3 & & P_{2,3,4} & \\
 & & P_{3,4} & & \\
x_4: & y_4 = P_4 & & &
\end{array}\right.$$

An improvement on the recurrence (3.1) is to keep track of the small differences between parents and daughters, namely to define for $m = 1, 2, ..., N-1$,

$$C_{m,i} \equiv P_{i,...,(i+m)} - P_{i,...,(i+m-1)} \tag{3.2}$$

$$D_{m,i} \equiv P_{i,...,(i+m)} - P_{(i+1),...,(i+m)} \tag{3.3}$$

Then one can easily derive from (3.1.1) the relations

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \tag{3.4}$$

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \tag{3.5}$$

At each level m, the C's and D's are the corrections that make the interpolation one order higher. The final answer $P_{1,...,N}$ is equal to the sum of any y_i plus a set of C's and/or D's that form a path through the family tree to the rightmost daughter.

3.2 Cubic Spline Interpolation

Given tabulated function $y_i = y(x_i), i = 1...N$, we shall focus attention on one particular interval, between x_j and x_{j+1}. Linear interpolation in that interval gives the interpolation formula

$$y = Ay_i + By_{i+1} \tag{3.6}$$

where

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j} \tag{3.7}$$

Since it is (piecewise) linear, equation (3.6) is continuous and has zero second derivative in the interior of each interval, but is not well suited for interpolating smooth (i.e., differentiable) functions. The goal of cubic spline interpolation is to get an interpolation formula that is smooth in the first derivative, and continuous in the second derivative, both within an interval and at its boundaries.
To define it more formally, we let $y(x)$ denote the piecewise cubic function on $[x_0, x_n]$, that is, there exists cubic $q_0(x), ..., q_n(x)$ such that $y(x) = q_k(x)$ on $[x_k, x_{k+1}]$. Besides y(x) interpolates $y(x_1), y(x_2), ..., y(x_n)$, therefore, $q_k(x) = y(x)$ on $[x_k, x_{k+1}]$ must satisfy

$$q_k(x_k) = y_k \text{ and } q_k(x_{k+1}) = y_{k+1} \text{ for } k = 0, 1, ..., n-1 \tag{3.8}$$

because $y(x)$ passes through $y(x_1), y(x_2), ..., y(x_n)$. Furthermore, we call $y(x)$ a cubic spline if the pieces $q_k(x_k)$ have the same slope and same concavity at the knots where they are joined, that is,

$$q'_{k-1}(x_k) = q'_k(x_k) = s'(x_k) \text{ for } k = 1, 2, ..., n-1 \tag{3.9}$$

$$q''_{k-1}(x_k) = q''(x_k) = s(x_k) \text{ for } k = 1, 2, ..., n-1 \tag{3.10}$$

The $2n$ conditions in (3.8), along with the $n-1$ conditions in each of (3.9) and (3.10) ensure that $y(x)$ and both its first and second derivatives are continuous in $[x_0, x_n]$. If y(x) is piecewise cubic on $[x_0, x_n]$, then its second derivative $y''(x)$ is piecewise linear on $[x_1, x_n]$; in particular, by (3.10), $q''_k(x)$ is linear and interpolates $(x_k, s''(x_k))$ and $(x_{k+1}, s''(x_{k+1}))$ in $[x_k, x_{k+1}]$. So

$$q''_k(x) = y''(x_k)(\frac{x - x_{k+1}}{x_k - x_{k+1}}) + y''_{k+1}(\frac{x - x_k}{x_{k+1} - x_k}) \text{ for } k = 0, 1, ..., n-1 \tag{3.11}$$

Integrating (3.11) twice with respect to x gives for $k = 0, 1, ..., n-1$,

$$q''_k(x) = \frac{y''(x_k)}{x_{k+1} - x_k}\frac{(x_{k+1} - x)^3}{6} + \frac{y''(x_{k+1})}{x_{k+1} - x_k}\frac{(x - x_k)^3}{6}$$

$$+A_k(x - x_k) + B_k(x_{k+1} - x) \tag{3.12}$$

where A_k and B_k are arbitrarily constants. Because $q_k(x)$ must satisfy (3.8), we can get for $k = 0, 1, ..., n-1$,

$$y_k = \frac{y''(x_k)}{6}(x_{k+1} - x_k)^2 + B_k(x_{k+1} - x_k)$$

and

$$y_{k+1} = \frac{y''(x_k)}{6}(x_{k+1} - x_k)^2 + A_k(x_{k+1} - x_k)$$

After solving A_k and B_k and substituting in (3.12), we obtain

$$q_k(x) = \frac{y''(x_k)}{6}[\frac{(x_{k+1} - x_k)^3}{x_{k+1} - x_k} - (x_{k+1} - x)(x_{k+1} - x_k)] + \frac{y''(x_{k+1})}{6}[\frac{(x - x_k)^3}{x_{k+1} - x_k} - (x_{k+1} - x_k)(x - x_k)]$$

$$+y_k[\frac{x_k - x}{x_{k+1} - x_k}] + y_{k+1}[\frac{x - x_k}{x_{k+1} - x_k}] \text{ for } k = 0, 1, ..., n - 1 \qquad (3.13)$$

Once we know the value of $y''(x_k)$ and $y''(x_{k+1})$, $q_k(x)$ is determined and can be used to evaluate $y(x)$ for $x_k < x < x_{k+1}$. Thus, we must find the second derivatives

$$y''(x_0), y''(x_1), y''(x_2), ..., y''(x_n)$$

To this end, we impose (3.9) and differentiate (3.13), which gives

$$q'_k(x) = \frac{y''(x_k)}{6}[\frac{-3(x_{k+1} - x)^2}{x_{k+1} - x_k}] + \frac{y''(x_{k+1})}{6}[\frac{3(x - x_k)^2}{x_k - x_{k+1}}] + \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \qquad (3.14)$$

Hence for $k = 0, 1, ..., n - 1$,

$$q'_k(x_k) = \frac{y''(x_k)}{6}[-2(x_{k+1} - x_k)] + \frac{y''(x_{k+1})}{6}[-(x_{k+1} - x_k)] + \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \qquad (3.15)$$

$$q'_k(x_{k+1}) = \frac{y''(x_k)}{6}[(x_{k+1} - x_k)] + \frac{y''(x_{k+1})}{6}[2(x_{k+1} - x_k)] + \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \qquad (3.16)$$

Replacing k by $k - 1$ in (3.16) to get $q'(x_k)$, and equating to (3.15) gives

$$(x_k - x_{k-1})y''(x_{k-1}) + 2(x_{k+1} - x_{k-1})y''(x_k) + (x_{k+1} - x_k)y''(x_{k+1}) =$$

$$6[\frac{y_{k+1} - y_k}{x_{k+1} - x_k}] - [\frac{y_k - y_{k-1}}{x_k - x_{k-1}}] \text{ for } k = 1, , ..., n - 1$$

There are $n+1$ unknowns, $y''(x_0), ..., y''(x_n)$. However, since it has only $n-1$ equation, it has infinitely many solutions. For a unique solution, we need to specify two further conditions. The most common way of doing this are either

1. set one or both of $y''(x_0)$ and $y''(x_n)$ to zero, giving the so-called natural spline.

2. set either of $y''(x_0)$ and $y''(x_n)$ to values calculated from equation (3.14) so as to make the first derivative of interpolating function have a specified value on either or both boundaries.

3.3 Interpolation in Two or More Dimensions

In two dimensions, we are given m points, $x_1, x_2, ..., x_m$, in the x dimension, n points, $y_1, y_2, ..., y_n$, in the y dimension, and $m * n$ functional values, $p(x_1, y_1), p(x_1, y_2)$ $, ..., p(x_2, y_n), ..., p(x_m, y_n)$ in the z dimension. Our goal is to estimate, by interpolation, the function p at some unknown point (x^*, y^*), where $x_i < x^* < x_{i+1}$, $y_j < y^* < y_{j+1}$ for $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$. The basic idea is to break up the problem into a succession of one-dimensional interpolations.

1. First, fixing $y = y_j$, we do n one-dimensional interpolations in the x direction by use of the points,(x_i, y_1) for $i = 1, 2, ..., m$, (x_i, y_2) for $i = 1, 2, ..., m$,..., and (x_i, y_n) for $i = 1, 2, ..., m$, respectively to get function values at the points (x^*, y_j) for $j = 1, 2, ..., m$.

2. Finally, we do a last interpolation in the y dimension to get the answer. Figure 3.1 illustrates our description graphically.

Three-dimension interpolation is analogous to two-dimensional interpolation in every way. Because the four dimension graph cannot be shown, we only plot its domain. Suppose we are given points, (x_i, y_j, z_k) for $i = 0, 1, 2$, $j = 0, 1, 2, 3$, and $k = 0, 1$.

1. First, fixing $y = y_j$ and $z = z_k$, we do 8 one-dimensional interpolations in the x direction by use of the points,(x_i, y_0, z_0) for $i = 0, 1, 2$, (x_i, y_1, z_0) for $i = 0, 1, 2$,..., and (x_i, y_3, z_1) for $i = 0, 1, 2$, respectively to get function values at the points (x^*, y_j, z_k) for $j = 0, 1, 2, 3$ and $k = 0, 1$. Figure 3.2 illustrates our description graphically.

2. Second, fixing $y = y_j$, we do 4 one-dimensional interpolations in the z direction by use of the points, (x^*, y_0, z_k) for $k = 0, 1$, (x^*, y_1, z_k) for $k = 0, 1$,..., and (x^*, y_3, z_k) for $k = 0, 1$, respectively to get function values at the points (x^*, y_j, z^*) for $j = 0, 1, 2, 3$. Figure 3.3 illustrates our description graphically.

3. Finally, we do a last interpolation in the y dimension to get the answer. Figure 3.4 illustrates our description graphically.

To define it more generally, suppose we are given $m * n * o$ points, (x_i, y_j, z_k) for $i = 1, ..., m$, $j = 1, ..., n$,and $k = 1, ..., o$, and $m{*}n{*}o$ functional values, $p(x_1, y_1, z_1), ..., p(x_m, y_n, z_o)$.

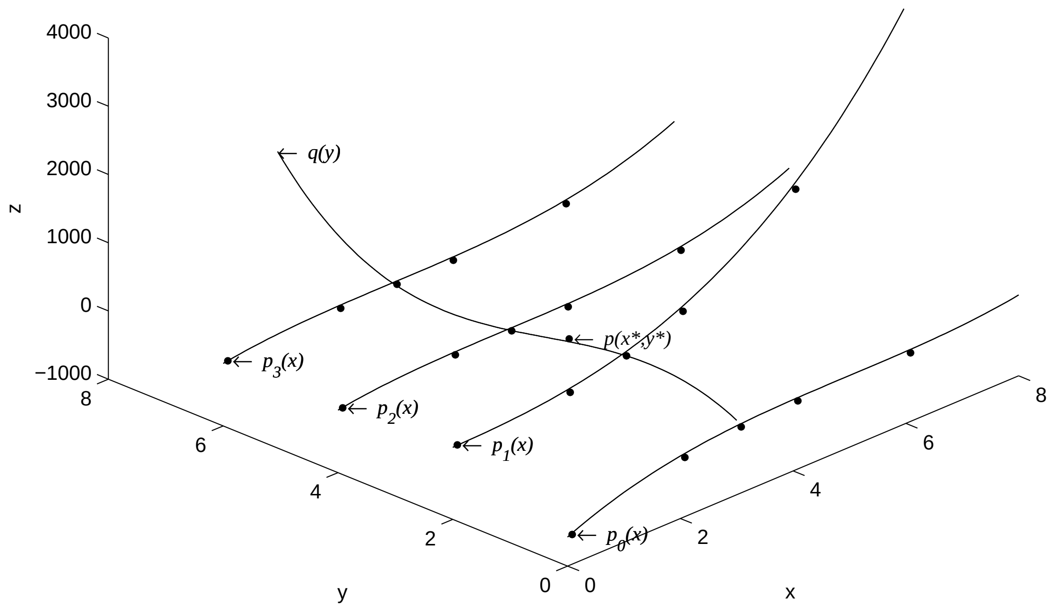

Figure 3.1: dim3

Our goal is to estimate, by interpolation, the function p at some unknown point (x^*, y^*, z^*), where $x_i < x^* < x_{i+1}$, $y_j < y^* < y_{j+1}$ and $z_k < z^* < z_{k+1}$.

1. First, fixing $y = y_j$ and $z = z_k$, we do $n * o$ one-dimensional interpolations in the y direction by use of the points,(x_i, y_1, z_1) for $i = 1, ..., m$, (x_i, y_2, z_1) for $i = 1, ..., m$,..., and (x_i, y_n, z_o) for $i = 0, 1, 2$, respectively to get function values at the points (x^*, y_j, z_k) for $j = 1, ..., n$ and $k = 1, ..., o$.

2. Second, fixing $y = y_j$, we do n one-dimensional interpolations in the z direction by use of the points, (x^*, y_1, z_k) for $k = 1, ..., o$, (x^*, y_2, z_k) for $k = 1, ..., o$,..., and (x^*, y_n, z_k) for $k = 1, ..., o$, respectively to get function values at the points (x^*, y_j, z^*) for $j = 1, ..., n$.

3. Finally, we do a last interpolation in the y dimension to get the answer.

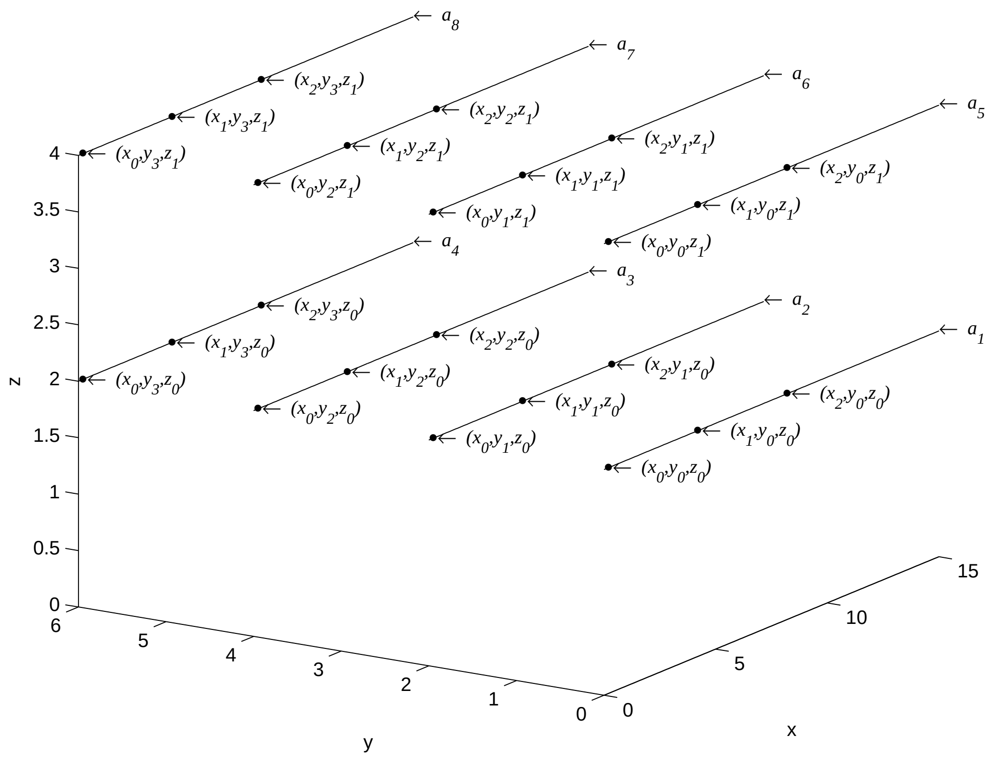

Figure 3.2: di1

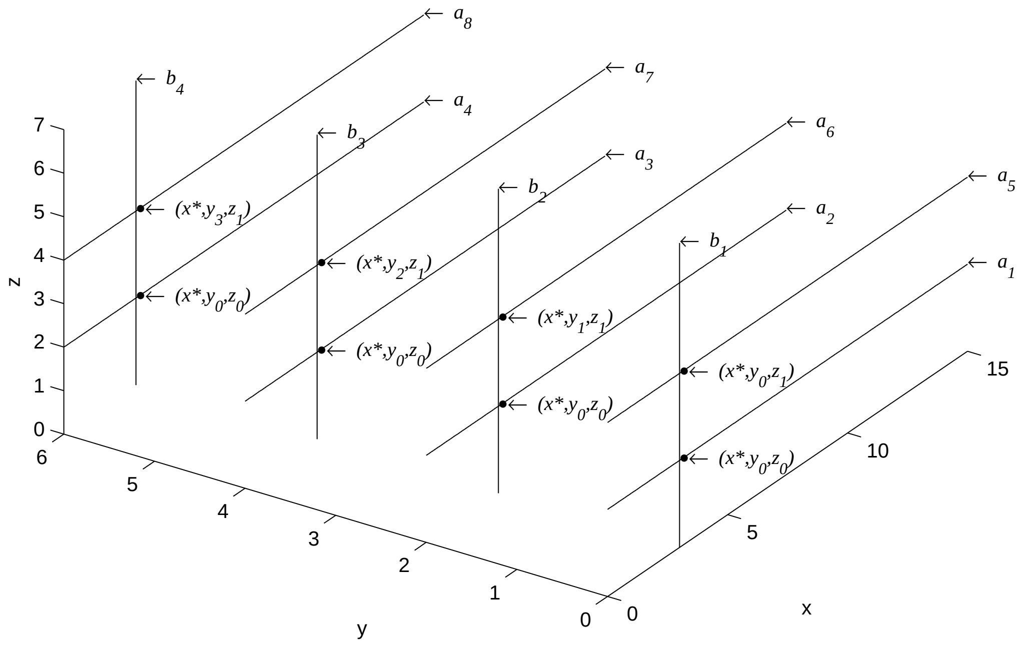

Figure 3.3: di2

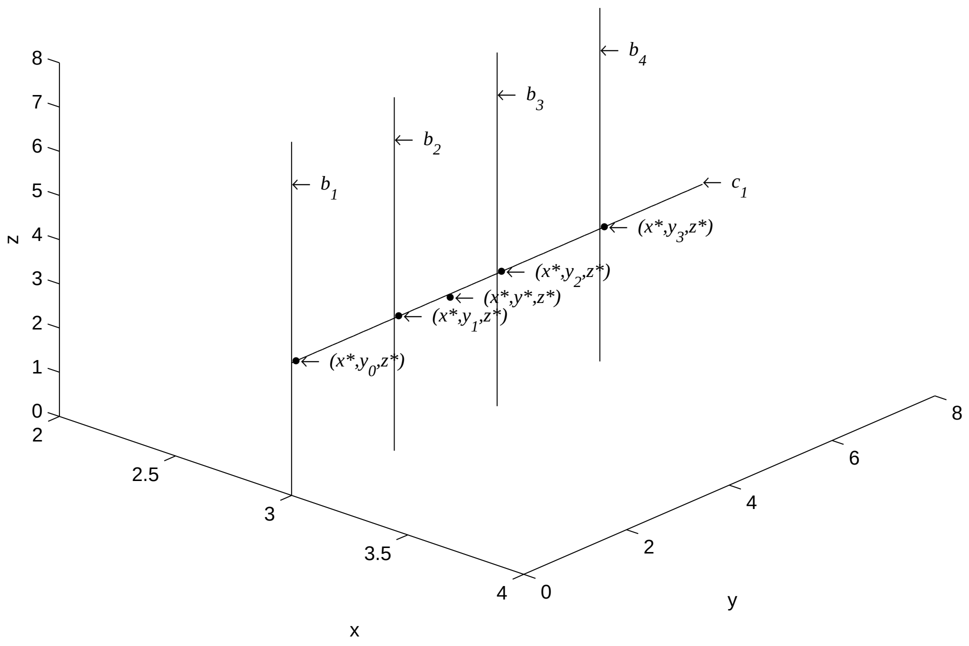

Figure 3.4: di3

3.4 Comparison Between Polynomial and Cubic Spline Interpolation

3.4.1 Computational complexity

In one dimensional interpolation with m points in the x dimension, the computational complexity of polynomial interpolation is $O(m^2)$ while that of the cubic spline interpolation is $O(m + log(m))$. For the cubic spline interpolation, we need $O(m)$ steps to calculate the second derivative, $y''(x_0), ..., y''(x_m)$, so that each piecewise cubic is determined. Besides, searching two second derivative $y''(x_k)$ and $y''(x_{k+1})$ to determine our desired cubic, $q_k(x)$ takes $O(log(m))$ time steps. In two dimensional interpolation with m points in the x dimension and n points in the y dimension, the computational complexity of polynomial interpolation is $O(n*m^2+n^2)$ while that of the cubic spline interpolation is $O(n*m+n*log(m)+n+log(n))$. This is because we need to determine n polynomials or splines in the y direction by use of m points in the x dimension each and get the interpolated value at (x^*, y_j) for $j = 1, 2, ..., m$, which takes $O(n*m^2)$ steps for the polynomial interpolation and $O(n*m+n*log(m))$ for the cubic spline interpolation. Then we need to determine one polynomial or spline in the x direction by use of n points in the y dimension and the interpolated point (x^*, y^*), which takes $O(n^2)$ steps for the polynomial interpolation and $O(n + log(n))$ steps. Adding computational complexity in the above steps may get our desired answer. In three dimensional interpolation with m points in the x dimension, n points in the y dimension and o points in the z dimension, the computational complexity of polynomial interpolation is $O(o*n*m^2+n*o^2+n^2)$ while that of the cubic spline interpolation is $O(o*n*m+o*n*log(m)+o*n+n*log(o)+n+log(n))$. This is because we need to determine $o*n$ polynomials or splines in the y direction by use of $m*o$ points in the $x-z$ dimension each and get the interpolated value at (x^*, y_j, z_k) for $j = 1, ..., n$ and $k = 1, ..., o$, which takes $O(o*n*m^2)$ steps for the polynomial interpolation and $O(o*n*m+o*n*log(m))$ for the cubic spline interpolation. Then we need to determine n polynomials or splines in the z direction by use of n points in the $y-z$ dimension and the interpolated point (x^*, y_j, z^*) for $j = 1, ..., n$, which takes $O(n*o^2)$ steps for the polynomial interpolation and $O(o*n+n*log(o))$ steps. Finally, we need to determine one polynomial or spline in the x direction by use of n points in the y dimension and the interpolated point (x^*, y^*, z^*), which takes $O(n^2)$ steps for the polynomial interpolation and $O(n+log(n))$ steps. Adding computational complexity in the above steps may get our desired answer.

3.4.2 Polynomial wiggle problem

For a smooth nonpolynomial curve, cubic splines interpolation is better than polynomial interpolation because polynomial interpolation is prone to wiggle between knots, which is the well-known *polynomial wiggle problem:*

Table 3.1: **Computational complexity**

dimension	Polynomial interpolation	Cubic spline interpolation
$1-dim$	$O(m^2)$	$O(m+logm)$
$2-dim$	$O(n*m^2+n^2)$	$O(n*m+n*logm+n+logn)$
$3-dim$	$O(o*n*m^2+n*o^2+n^2)$	$O(o*n*m+o*n*logm+o*n+n*logo+n+logn)$

If points $P_1, P_2, ..., P_m$ do not actually comes from a polynomial curve, then an attempt to let a polynomial $p(x)$ go through them will cause $p(x)$ to have oscillations between successive $P_k's$. These oscillations get larger as the degree of $p(x)$ is allowed to increase.

For example, having points $P_1, ..., P_6$, coming form $y = 10e^{-\frac{2}{x}} - 4$, we want to find a polynomial passing through them. Figure 3.1 illustrates the Polynomial wiggle problem and figure 3.6 is the true curve passing through them. We can see that curve in figure 3.5 oscillates greatly and bears no resemblance to the true curve in figure 3.6.
The recursive formula of the American put option,

$$P(i,j) = \max(\frac{pP(i+1,j+1)+(1-p)P(i+1,j)}{R}, X - S_0u^jd^{i-j})$$

where

$$u = e^{\sigma dt^{0.5}}, d = e^{-\sigma dt^{0.5}}, R = e^{rdt},$$

is a smooth nonpolynomial function before the exercise boundary. Therefore, since the the value of an American put option is a function of e^σ and e^r , interpolating more points in the σ and r dimension may not only fail to increase the accuracy but also cause tremendous error in the polynomial interpolation. However, interpolating more points in Cubic spline interpolation will improve its accuracy. The numerical results will be shown in the next chapter.

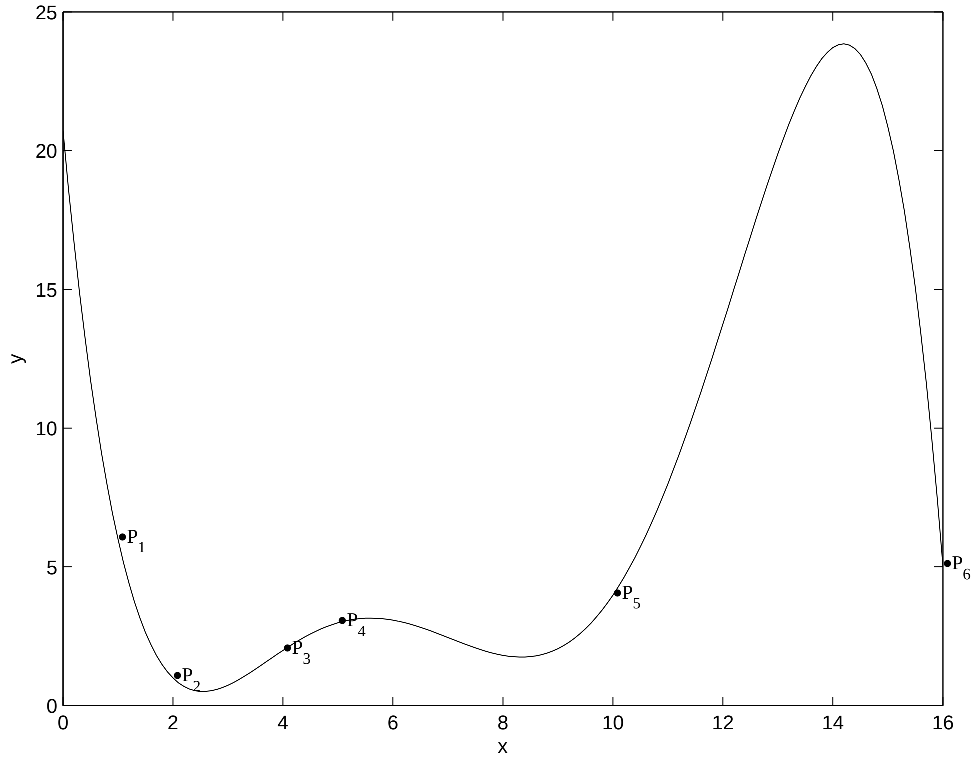

Figure 3.5: Polynomial wiggle problem

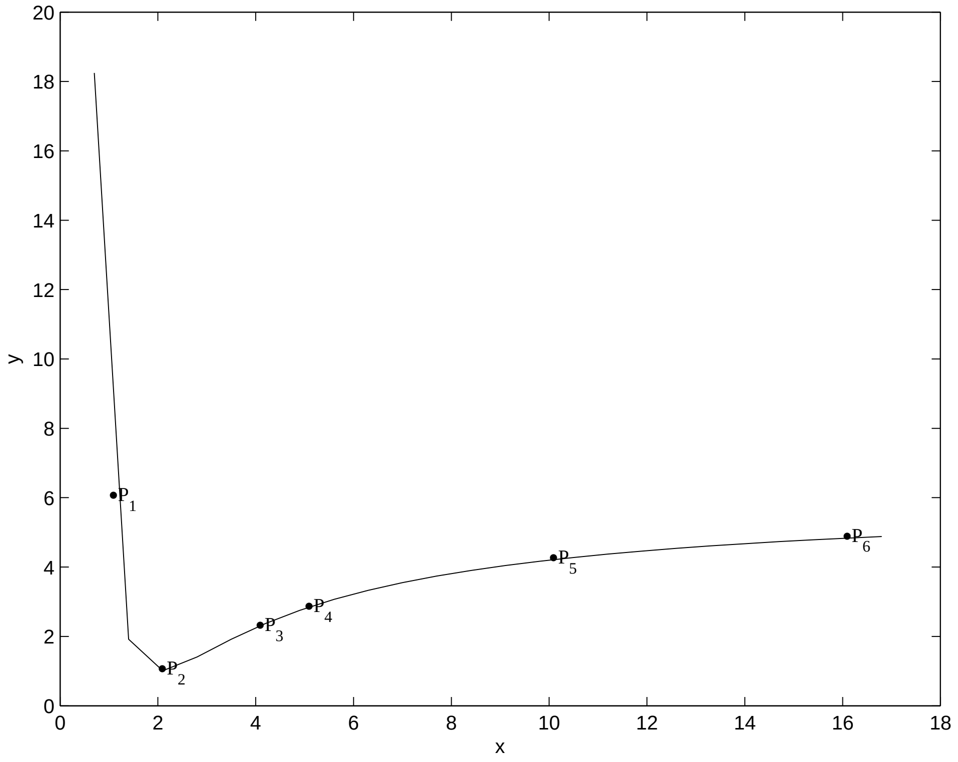

Figure 3.6: The curve that give rise to $P_1, ..., P_6$.

Chapter 4

Numerical Results

4.1 Building table

Before we want to get the price of an American option fast and accurately, we need to build the table storing the precomputed results of American options first. This section describes how we build this table.
To build the look-up table of option prices, first we need to consider the parametrisation we used. An American put option is a function of underlying stock price(S_t), exercise price(K), risk-free interest rate(r), volatility of underlying stock price(σ), and time to maturity(T), which can be expressed as,

$$P(S_t, K, r, \sigma, T).$$

Because the exercise price is fixed and tomorrow's time to maturity is known, we only have to partition and build a table in three dimensions, stock price(S_0), risk-free interest rate(r), and volatility of underlying stock price(σ). We let the number of partition in S_0, r, and σ dimension be m, n, and o, respectively and the ranges in each dimension we choose to partition are

$$S_{t-1}: \quad [1.07xS_{t-1}, 0.93xS_{t-1}] \quad r: \quad [0.1, 0] \quad \sigma^2: \quad [2x\sigma^2, 0.5x\sigma^2]$$

We use relative error and absolute error to evaluate the accuracy of interpolation. Relative error is defined as

$$RE = \frac{p - \hat{p}}{p}$$

, and absolute error is defined as

$$AE = |\frac{p - \hat{p}}{p}|$$

where p is 100-step Binomial price and $\hat{p}$ is look-up table price.

4.2 numerical results

This section introduces the results of interpolating the price of an American option in the table we built in the prior section by the method of polynomial and cubic spline interpolation. Table (4.1) and (4.2) shows the general cases in interpolating an American option based on polynomial and cubic spline interpolation, respectively. Table (4.3) and (4.5) shows the *polynomial wiggle problem* arising from interpolating a nonpolynomial function with a polynomial function. Table (4.4) and (4.6) shows that the results of cubic spline interpolation are still acceptable no matter how fine we partition. Table (4.7) and (4.8) shows the results of interpolating the price of an American option straddling the exercise boundary. Table (4.9) and (4.10) shows the results after we partition the grid finer so as to lower the error causing by interpolating the price of an American option straddling the exercise boundary.

Table 4.1: **Polynomial interpolation**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4800	0.5	20	2	363.322	364.464	-0.003144	1.142460
5000	4850	0.5	20	2	334.678	334.144	0.001595	0.533864
5000	4900	0.5	20	2	308.873	308.82	0.000171	0.052936
5000	4950	0.5	20	2	284.213	284.342	-0.000454	0.129088
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000458
5000	5050	0.5	20	2	238.83	238.945	-0.000481	0.114774
5000	5100	0.5	20	2	218.277	218.231	0.000209	0.045603
5000	5150	0.5	20	2	198.202	197.901	0.001517	0.300750
5000	5200	0.5	20	2	181.331	182.398	-0.005885	1.067120
5000	5000	0.5	20	1	269.989	270.011	-0.000083	0.0224928
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000457825
5000	5000	0.5	20	3	250.098	250.096	0.000009	0.00223362
5000	5000	0.5	20	4	241.03	241.03	-0.000001	0.000139438
5000	5000	0.5	20	5	232.46	232.46	0.000002	0.000479013
5000	5000	0.5	20	6	224.337	224.337	-0.000001	0.000211508
5000	5000	0.5	20	7	216.627	216.627	0.000001	0.000261474
5000	5000	0.5	20	8	209.284	209.284	0.000002	0.000359369
5000	5000	0.5	20	9	202.31	202.291	0.000094	0.0190052
5000	5000	0.5	15	2	190.211	190.212	-0.000006	0.001107
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000458
5000	5000	0.5	25	2	329.283	329.283	-0.000001	0.000351
5000	5000	0.5	30	2	398.822	398.822	0.000000	0.000013
5000	5000	0.5	35	2	468.277	468.277	0.000001	0.000420

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.000280389
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=0.001472845
Mean absolute relative error($\frac{1}{n}\sum |p_i - \hat{p}_i|$)=0.149349637
Maximum relative error(max$\frac{(p_i-\hat{p}_i)}{p_i}$)=0.00588493
Maximum absolute error(max$|p_i - \hat{p}_i|$)=1.142460
$S_{t-1} = 5000$ and S is tomorrow's stock price
$m = 21, n = 16$, and $o = 16$

Table 4.2: **Cubic spline interpolation**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4800	0.5	20	2	363.322	363.348	-0.000072	0.0261708
5000	4850	0.5	20	2	334.678	334.496	0.000542	0.181521
5000	4900	0.5	20	2	308.873	308.893	-0.000064	0.0197189
5000	4950	0.5	20	2	284.213	284.256	-0.000152	0.0430896
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000457825
5000	5050	0.5	20	2	238.83	238.86	-0.000125	0.0298431
5000	5100	0.5	20	2	218.277	218.303	-0.000118	0.0257643
5000	5150	0.5	20	2	198.202	198.239	-0.000189	0.037442
5000	5200	0.5	20	2	181.331	181.331	0.000001	0.000238648
5000	5000	0.5	20	1	269.989	269.979	0.000035	0.00950098
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000457825
5000	5000	0.5	20	3	250.098	250.097	0.000005	0.00129289
5000	5000	0.5	20	4	241.03	241.03	-0.000001	0.000139438
5000	5000	0.5	20	5	232.46	232.459	0.000002	0.000524934
5000	5000	0.5	20	6	224.337	224.337	-0.000001	0.000211508
5000	5000	0.5	20	7	216.627	216.626	0.000005	0.00101464
5000	5000	0.5	20	8	209.284	209.284	0.000002	0.000359369
5000	5000	0.5	20	9	202.31	202.307	0.000014	0.0028402
5000	5000	0.5	15	2	190.211	190.212	-0.000007	0.001417
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000458
5000	5000	0.5	25	2	329.283	329.283	-0.000001	0.000373
5000	5000	0.5	30	2	398.822	398.822	0.000000	0.000013
5000	5000	0.5	35	2	468.277	468.277	0.000000	0.000081

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p_i})$)=-0.000005
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p_i})^2}$)=0.000130753
Mean absolute relative error($\frac{1}{n}\sum |p_i - \hat{p_i}|$)=0.01664915
Maximum relative error($\max\frac{(p_i - \hat{p_i})}{p_i}$)=0.000542
Maximum absolute error($\max|p_i - \hat{p_i}|$)=0.181521
$S_{t-1} = 5000$ and S is tomorrow's stock price
$m = 21, n = 16$, and $o = 16$
Table (4.1) and (4.2) are the results of interpolating 21 points in the S dimension, 16 points in the σ^2 dimension, and 16 points in r dimension based on polynomial and cubic spline interpolation. We can see that holding other parameters the same as tomorrow's stock price is more away from today's stock price, 5000, the error tends to become larger in both interpolation method, which is just coincidence. If the error is not acceptable, the most simple way we can do is to increase the node we partition in S dimension. We can also see that holding other parameters the same, to change tomorrow's σ^2 or r causes very small error. Likewise, we can increase the partition for smaller error.

Table 4.3: **Polynomial interpolation with wiggle problem in the r dimension**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4800	0.5	20	2	363.322	2040.7	-4.61679	1677.38
5000	4850	0.5	20	2	334.678	194.062	0.420152	140.616
5000	4900	0.5	20	2	308.873	-1242.58	5.02295	1551.45
5000	4950	0.5	20	2	284.213	270.965	0.0466145	13.2485
5000	5000	0.5	20	2	259.715	-490.134	2.8872	749.849
5000	5050	0.5	20	2	238.83	-1085.29	5.5442	1324.12
5000	5100	0.5	20	2	218.277	1322.85	-5.0604	1104.57
5000	5150	0.5	20	2	198.202	-14.7026	1.07418	212.905
5000	5200	0.5	20	2	181.331	1942.79	-9.71403	1761.46

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.488435944

Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=4.794921438

Mean absolute relative error($\frac{1}{n}\sum|p_i - \hat{p}_i|$)=948.3998333

Maximum relative error($\max\frac{(p_i-\hat{p}_i)}{p_i}$)=9.71403

Maximum absolute error($\max|p_i - \hat{p}_i|$)=1761.46

$S_{t-1} = 5000$ and S is tomorrow's stock price

$m = 21, n = 16$, and $o = 100$

Table 4.4: **Cubic spline interpolation without wiggle problem in the r dimension**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4800	0.5	20	2	363.322	363.349	-0.000074	0.026723
5000	4850	0.5	20	2	334.678	334.496	0.000543	0.181572
5000	4900	0.5	20	2	308.873	308.893	-0.000064	0.0197623
5000	4950	0.5	20	2	284.213	284.256	-0.000152	0.0431654
5000	5000	0.5	20	2	259.715	259.714	0.000002	0.000619204
5000	5050	0.5	20	2	238.83	238.86	-0.000125	0.0298814
5000	5100	0.5	20	2	218.277	218.303	-0.000119	0.0258755
5000	5150	0.5	20	2	198.202	198.24	-0.000190	0.0375936
5000	5200	0.5	20	2	181.331	181.331	0.000002	0.000324378

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.000020

Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=2.088586E-04

Mean absolute relative error($\frac{1}{n}\sum|p_i - \hat{p}_i|$)=0.040613

Maximum relative error($\max\frac{(p_i-\hat{p}_i)}{p_i}$)=0.000543

Maximum absolute error($\max|p_i - \hat{p}_i|$)=0.181572

$S_{t-1} = 5000$ and S is tomorrow's stock price

$m = 21, n = 16$, and $o = 100$

Table 4.5: **Polynomial interpolation with wiggle problem in the σ^2 dimension**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4800	0.5	20	2	363.322	1727.57	-3.75492	1364.24
5000	4850	0.5	20	2	334.678	468.981	-0.40129	134.303
5000	4900	0.5	20	2	308.873	418.244	-0.354099	109.371
5000	4950	0.5	20	2	284.213	156.844	0.448145	127.369
5000	5000	0.5	20	2	259.715	268.458	-0.0336627	8.7427
5000	5050	0.5	20	2	238.83	308.644	-0.292315	69.8135
5000	5100	0.5	20	2	218.277	120.177	0.44943	98.1002
5000	5150	0.5	20	2	198.202	-1317.77	7.64861	1515.97
5000	5200	0.5	20	2	181.331	-6207.71	35.2341	6389.04

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=4.327110922
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=12.0868077
Mean absolute relative error($\frac{1}{n}\sum|p_i - \hat{p}_i|$)=1090.772156
Maximum relative error($\max\frac{(p_i-\hat{p}_i)}{p_i}$)=35.2341
Maximum absolute error($\max|p_i - \hat{p}_i|$)=6389.04
$S_{t-1} = 5000$ and S is tomorrow's stock price
$m = 21, n = 260$, and $o = 16$

Table 4.6: **Cubic spline interpolation without wiggle problem in the σ^2 dimension**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4800	0.5	20	2	363.322	363.348	-0.000073	0.026415
5000	4850	0.5	20	2	334.678	334.495	0.000546	0.182720
5000	4900	0.5	20	2	308.873	308.893	-0.000064	0.019845
5000	4950	0.5	20	2	284.213	284.256	-0.000151	0.043022
5000	5000	0.5	20	2	259.715	259.715	0.000002	0.000458
5000	5050	0.5	20	2	238.83	238.86	-0.000125	0.029844
5000	5100	0.5	20	2	218.277	218.303	-0.000118	0.025763
5000	5150	0.5	20	2	198.202	198.239	-0.000189	0.037427
5000	5200	0.5	20	2	181.331	181.331	0.000001	0.000253

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.000019
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=0.000209658
Mean absolute relative error($\frac{1}{n}\sum|p_i - \hat{p}_i|$)=0.040638655
Maximum relative error(max$\frac{(p_i - \hat{p}_i)}{p_i}$)=0.000545959
Maximum absolute error(max$|p_i - \hat{p}_i|$)=0.182720
$S_{t-1} = 5000$ and S is tomorrow's stock price
$m = 21, n = 260$, and $o = 16$
However, If we try to increase the partition in the σ^2 or the r dimension, for example, we increase the points in r dimension in table (4.3) and (4.4) to 100 and σ^2 dimension in table (4.5) and (4.6)to 260 , respectively. In table (4.3) and (4.5) we can see the *polynomial wiggle problem* occurs and the results are completely unacceptable, while in table (4.4) and (4.6) we can see that interpolating more points in the closed interval by cubic spline still makes the error acceptable, though it does not improve accuracy. Therefore, it is very important to optimize the number of points partitioned to prevent *polynomial wiggle problem* for polynomial interpolation. Cubic spline interpolation is more preferable in this respect because we can partition these three dimensions as finer as we want. The reason why table (4.4) and (4.6) does not improve the accuracy compared with table (4.2) lies in that the majority of the error comes form the S dimension.

Table 4.7: **Polynomial interpolation straddling the exercise boundary**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4178	0.25	20	2	822	822.019	-0.000024	0.019499
5000	4180	0.25	20	2	820	820.027	-0.000033	0.026761
5000	4182	0.25	20	2	818	818.035	-0.000043	0.035476
5000	4182.5	0.25	20	2	817.5	817.538	-0.000046	0.037898
5000	4183	0.25	20	2	817.0022	817.04	-0.000047	0.038222
5000	4184	0.25	20	2	816.011	816.046	-4.26E-05	0.0347828
5000	4186	0.25	20	2	814.03	814.058	-3.42E-05	0.0278103

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.000039
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=0.000039
Mean absolute relative error($\frac{1}{n}\sum |p_i - \hat{p}_i|$)=0.031492671
Maximum relative error($\max\frac{(p_i - \hat{p}_i)}{p_i}$)=0.000047
Maximum absolute error($\max|p_i - \hat{p}_i|$)=0.220449
$S_{t-1} = 4000$ and S is tomorrow's stock price
$m = 21, n = 16$, and $o = 16$

Table 4.8: **Cubic spline interpolation straddling the exercise boundary**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4178	0.25	20	2	822	822.026	-0.000031	0.025683
5000	4180	0.25	20	2	820	820.034	-0.000041	0.033825
5000	4182	0.25	20	2	818	818.043	-0.000053	0.043135
5000	4182.5	0.25	20	2	817.5	817.546	-0.000056	0.045654
5000	4183	0.25	20	2	817.0022	817.048	-0.000056	0.046051
5000	4184	0.25	20	2	816.011	816.054	-0.000052	0.042686
5000	4186	0.25	20	2	814.03	814.066	-0.000044	0.035549

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.000048
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=0.000048
Mean absolute relative error($\frac{1}{n}\sum |p_i - \hat{p}_i|$)=0.038940
Maximum relative error($\max\frac{(p_i - \hat{p}_i)}{p_i}$)=0.000056
Maximum absolute error($\max|p_i - \hat{p}_i|$)=0.046051
$S_{t-1} = 4000$ and S is tomorrow's stock price
$m = 21, n = 16$, and $o = 16$

Table 4.9: **Polynomial interpolation with finer grid**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4178	0.25	20	2	822	822	0.000000	0.000036
5000	4180	0.25	20	2	820	820	0.000000	0.000118
5000	4182	0.25	20	2	818	818	0.000000	0.000000
5000	4182.5	0.25	20	2	817.5	817.501	-0.000001	0.001095
5000	4183	0.25	20	2	817.0022	817.003	-0.000001	0.001080
5000	4184	0.25	20	2	816.011	816.011	0.000000	0.000091
5000	4186	0.25	20	2	814.03	814.03	0.000000	0.000038

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.0000004
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=0.000001
Mean absolute relative error($\frac{1}{n}\sum |p_i - \hat{p}_i|$)=0.000351136
Maximum relative error(max$\frac{(p_i - \hat{p}_i)}{p_i}$)=0.000001
Maximum absolute error(max$|p_i - \hat{p}_i|$)=0.001095
$S_{t-1} = 4000$ and S is tomorrow's stock price
$m = 21, n = 16$, and $o = 16$

Table 4.10: **Cubic spline interpolation with finer grid**

K	S	T	σ^2	r	Binomial	Look-Up	Rel. Err.	Abs. Err.
5000	4178	0.25	20	2	822	822	0.000000	0.000007
5000	4180	0.25	20	2	820	820	0.000000	0.000090
5000	4182	0.25	20	2	818	818	0.000000	0.000000
5000	4182.5	0.25	20	2	817.5	817.501	-0.000001	0.001174
5000	4183	0.25	20	2	817.0022	817.003	-0.000001	0.001142
5000	4184	0.25	20	2	816.011	816.011	0.000000	0.000087
5000	4186	0.25	20	2	814.03	814.03	0.000000	0.000088

Means square error($\frac{1}{n}\Sigma(p_i - \hat{p}_i)$)=-0.0000004
Root mean square error($\sqrt{\frac{1}{n}\Sigma(p_i - \hat{p}_i)^2}$)=0.000001
Mean absolute relative error($\frac{1}{n}\sum|p_i - \hat{p}_i|$)=0.000370
Maximum relative error(max$\frac{(p_i-\hat{p}_i)}{p_i}$)=0.000001
Maximum absolute error(max$|p_i - \hat{p}_i|$)=0.001174
$S_{t-1} = 4000$ and S is tomorrow's stock price
$m = 21, n = 16$, and $o = 16$
Interpolating in cubes that straddle the exercise boundary causes larger error as we can see in table (4.7) and (4.8). Our remedy for this problem is to keep track of the cubes straddling the exercise boundary and partition them finer. In table (4.9) and (4.10) we can see that the error becomes smaller. However, we should be careful that *polynomial wiggle problem* may happen because of finer grid for the polynomial interpolation.

Chapter 5

Conclusion

The results in this thesis presents the method of look-up table to price American options. Polynomial interpolation and cubic spline interpolation are used to maintain its accuracy. One key factor in maintaining its accuracy depends on the number of partitions chosen. For the polynomial interpolation we should choose the number of partition properly so as to prevent the *polynomial wiggle problem*. Therefore, cubic spline interpolation is preferable because it becomes more accurate as the grid turns finer. The other key factor lies in that price function changes very rapidly at the exercise boundary. Therefore, keeping track of the cubes straddling the exercise boundary and partitioning the cubes finer are required to lower the error of interpolation.

Bibliography

[1] Lee W. Johnson and R. Dean Riess. *Numerical Analysis.* Addison-Wesley, 1982.

[2] Melvin J. and Maron. *Numerical Analysis: A Pratical Approach.* Macmillan, 1982.

[3] William H. Press., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery *Numerical Recipes in C.* Cambridge, 1992.

[4] L.C.G.Rogers and D. Taylay. *Numerical Methods in Finance.* Prentice-Hall, 2000.

[5] Hull, John. *Options, Futures, and Other Derivatives.* 4th edition. Cambridge, 1997.

[6] Yuh-Dauh Lyuu "Financial Engineering and Computation." Cambridge, 2002.