파생상품 C++

이종희 ◦ 김성기 ◦ 정다래 ◦ 김준석

2010-12-16

머리말

이 책에서는 금융에 관한 공식을 계산하는 방법에 대해서 C++을 이용해서 나타내고 있다. 적지 않은 사람들은 C++이 나온지 오래되었고 쓰기에도 불편하고 복잡하다고 생각한다. 그에 반해 MATLAB이나 spreadsheet는 쓰기에도 편하고 이것들이면 모든 계산을 다 할 수 있다고 이야기 한다. 하지만 C++의 경우 class의 개념을 통한 캡슐화가 가능하여 복잡한 계산도 부분 부분으로 나누어 쉽게 계산이 가능하고 효율적인 메모리 관리를 통해서 계산의 신속성을 높일 수 있는 등 C++의 장점은 쓰기에 불편하고 나온지 오래 되었다는 단점을 가리고도 남는다.

이러한 장점에도 아직까지 C++을 이용하여 실제적인 금융모델에 활용된 코드들을 배울수 있는 책이 상대적으로 적은 것이 사실이다. 이 책에서는 여러 가지 파생금융상품 중에서 옵션의 기본적인 가격결정모델과 모델의 수치해석을 다룬다. 초보자들에게 옵션거래의 핵심사항들을 간단명료하게 해설하고 C++ 코드를 제시함으로써 기본적인 수치해석 원리와 기법을 쉽고 정확하게 이해 할 수 있도록 했다. 옵션에 대해서 기본적인 일반상식, 경제학, 수학, 수치해석의 네 분야를 한권의 책으로 만들었다. 이 책이 스톡옵션의 가격결정을 위한 수치해석 기법을 공부하는 사람들의 입문서로서 도움이 되었으면 한다.

본 저서의 중점은 제 8장, 9장, 10장의 유한 차분법, Tree 가격결정모형, 그리고 몬테카를 시뮬레이션등 수치해석이나 내용의 완전성을 유지하기 위해 다른 주제들도 함께 담았다. 부족한 내용들은 서점에 나와 있는 다른 참고 도서를 참조하면 될 것이라 사료된다. 이 책이 파생상품 모델링에 대한 수치해석에 관심 있는 독자들에게 조그마한 보탬이라도 될 수 있기를

차 례

제 1 장

서론

본 저서는 서론을 제외하고 다음과 같이 총 9장으로 구성되어 있다.

제 2장은 파생금융상품 중의 하나인 KOSPI200지수 옵션에 대해서 다루었다.

제 3장은 MATLAB을 잘 모르는 독자들을 위해 MATLAB의 기본 사용법과 본 저서의 MATLAB 코드를 이해하기 위해서 필수적으로 알아야 하는 명령어들의 설명이 있다.

제 4장은 C++을 잘 모르는 독자들을 위해 C++컴파일러의 기본 사용법과 C++ 언어의 기본적인 개념들과 본 저서의 C++ 프로그래밍을 이해하기 위해서 필수적인 언어들의 설명이 있다.

제 5장에서는 기하 브라운 운동을 소개하며 Itô 보조정리를 유도한다.

제 6장은 옵션가격을 결정 하는 방정식인 Black-Scholes 편미분 방정식을 유도하며 열방정식의 해석해로부터 Black-Scholes 방정식의 해석해를 유도한다.

제 2 장

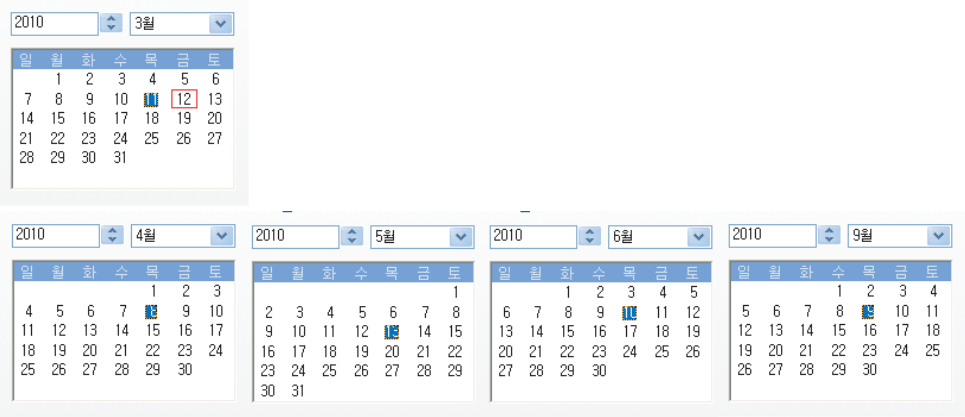
파생금융상품 (Derivatives)

파생금융상품은 기초자산(underlying assets)의 가격변동으로 인한 손실위험을 제거하기 위해 탄생했다. 대표적 파생금융상품으로는 선물, 옵션, 스왑 등이 있으며, 이 책에서는 이러한 상품 중 옵션에 대하여 다루기로 한다. 옵션계약의 대상이 되는 상품을 **기초자산**이라 하는데, 이는 크게 곡물, 축산물, 귀금속 등과 같은 일반적인 상품(commodity)과 주식, 채권, 주가지수(KOSPI 200), 통화 등과 같은 금융상품으로 구분할 수 있다. 각각의 옵션을 거래대상에 따라 상품 옵션(commodity option) 그리고 금융옵션(financial option)으로 나눌 수 있다.

제 1 절 옵션의 개념과 주요 용어

옵션(option)이란 특정 자산을 미리 정해진 가격(행사가격, strike price)으로 미래의 일정한 시기(만기, maturity)에 또는 일정한 기간 내에 매수 또는 매도할 권리가 내재된 계약을 일컫는다. 이를 유러피언 옵션(European option) 또는 아메리칸 옵션(American option)이라 한다. 이 때 특정자산을 매수할 권리를 **콜옵션**(call option), 매도할 수 있는 권리를 **풋옵션**(put option)이라고 한다.

그리고 새로이 2010년 9월 만기의 옵션이 상장되어 거래된다.



만기일이 가장 가까운 상품을 **근월물**이라 하고 그 다음이 만기인 순으로 **차월물**, **차차월물**이라고 한다.

제 4 절 옵션의 상태

유러피언 콜옵션의 경우 만기일에 기초자산의 시장가격이 행사가격보다 낮은 경우 옵션은 옵션매입자는 옵션권리의 취득에 의해 옵션 매도자에게 정해진 기간에 옵션계약의 내용에 대한 이행을 청구할 수도 있으며, 반대로 자신에게 불리한 경우 옵션계약을 포기할 수 있다. 가치를 지니지 못하게 되므로 옵션은 행사되지 않고 소멸되고 만다. 이와 같이 옵션의 권리를 행사하는 것이 이득이 되지 않는 상태를 외가격(out-of-the-money; OTM)상태라 하고 이 때의 옵션을 **외가격옵션**이라 부른다.

옵션의 행사가격이 기초자산의 시장가격과 동일하여 옵션을 행사하거나 행사하지 않는 경우가 무차별한 경우를 등가격상태라 하며 이러한 옵션을 **등가격옵션**(at-the-money option; ATM option)이라 한다. 한편, 시장에서 기초자산의 가격상황이 옵션을 행사하는 것이 유리한 경우, 예를 들어 콜옵션의 경우에는 기초자산의 시장가격이 행사가격보다 높은 상태를 내가격상태라 하며 이러한 옵션을 **내가격옵션**(in-the-money option; ITM option)이라 한다.

6.2 호가단위

1계약의 크기는 지수 1포인트당 10만원이며 호가가격 변동폭은 옵션가격이 3포인트 이상일 경우는 0.05포인트, 즉 5,000원이고 3포인트 미만일 경우는 1,000원, 즉 0.01포인트이다. 옵션거래는 가격제한폭이 없지만 시장안정을 위해서 호가가격이 전일의 대상자산 가격대비 ±15%를 벗어나는 경우 거래소에서 호가접수를 거부하는 호가한도 가격제도를 두고 있다.

6.3 거래시간

우리나라 주가지수 옵션거래는 주식시장의 개장 시간인 9시에 거래를 시작하며 주식시장 종료시점보다 15분 늦은 3시 15분에 거래가 종료된다. 이는 확정된 주가 지수에 따라 옵션 포지션을 조정할 수 있도록 하기 위함이다. 그러나 결제일에는 주식시장보다 10분 일찍 2시 50분에 거래를 종료하도록 되어 있다. 이러한 이유는 주식시장이 종료되기 전까지의 10분간 거래가격의 왜곡가능성을 배재하기 위해서이다. 근월물의 결제일의 옵션은 2시 50분에 매매가 종료되지만 다음월물의 거래는 계속 지속되며 공휴일의 경우는 거래가 되지 않는다.

6.4 거래량

매수호가와 매도호가가 일치하면 거래가 체결된다. A가 10계약 매수주문을 내고 동일한 금액으로 B가 30계약 매도주문을 내게 되면, 주문이 일치된 10계약이 이루어지는데 이 때 거래가 체결된 10계약이 거래량이 된다.

제 7 절 매매제도

7.1 행사가격

행사가격은 전날의 KOSPI 200 종가에 가장 가까운 행사가격과 2.5포인트씩 높인 행사가격 4개 그리고 2.5포인트씩 낮춘 행사가격 4개를 제시함으로써 모두 9개 행사가격을 가진 옵션이 거래된다. 단, 3, 6, 9, 12월이 결제일인 옵션은 5포인트 간격으로 5개의 행사가격이 설정된다.

7.4.1 사이드카(Sidecar)

사이드카는 프로그램 매매호가 관리제도의 일종으로, 주가지수선물시장에서 선물 가격이 급등락할 경우 선물시장의 충격이 현물 시장에 파급되는 것을 완화시키기 위하여 일시적으로 프로그램매매의 호가효력을 중단시키는 것이다. 선물의 거래중지와 동시에 옵션의 거래도 중지된다.

우리나라에서 사이드카의 발동요건은 KOSPI선물시장의 경우, 전날의 거래량이 가장 많았던 종목의 선물가격이 기준가격(전일종가) 대비 6% 이상 상승하거나 하락한 상태에서 코스닥스타 현물지수(선물거래대상지수)도 3% 이상 같은 방향으로 변동해 1분간 지속될 때 발동된다. 사이드카가 발동되면 주식시장의 프로그램 매매 호가가 5분간 효력이 정지된다. 5분 후에는 정상거래가 되며, 사이드카는 하루에 한 번만 쓸 수 있다. 또 주식시장 매매거래 종료 40분 전 이후, 즉 오후 2시 20분 이후에는 발동되지 않는다. 종전에는 선물가격이 6%이상 등락한 상태에서 1분간 지속될 경우 사이드카가 발동되었지만, 잦은 비현실적 경보음으로 오히려 시장을 왜곡한다는 비판을 받아온 코스닥시장의 사이드카 발동요건이 2009년 7월 6일 월요일을 기점으로 대폭 강화되었다.

7.4.2 서킷브레이커(Circuit Breaker)

종합주가지수가 직전 매매거래일보다 10% 이상 하락하여 1분간 지속되는 경우에는 주식시장의 모든 매매거래를 중단시키며 옵션의 거래도 중지되는 제도를 **서킷브레이커**라고 한다. 서킷 브레이커가 발동되면 20분 동안 시장 내 호가접수와 채권시장을 제외한 현물시장과 연계된 선물·옵션시장도 호가접수 및 매매거래를 중단한다. 서킷 브레이커가 발동된 후 20분이 지나면 매매거래를 재개되는데 이 때 시작가격은 재개시점부터 10분 동안 호가를 접수하여 단일가로 매매를 체결하여 거래가 다시 시작된다. 사이드카와 마찬가지로 하루에 1회만 발동할 수 있고, 장 종료 40분전 이후에는 발동하지 않는다.

또한 매매종료 후 위탁증거금에서 손실을 차감한 금액이 옵션의 미결제약정에 대한 유지증거금에 미달하는 경우 그 차액을 다음날 12시까지 증권회사에 추가로 납부해야 한다. 만약 기한까지 납부하지 않을 경우 증권회사는 임의로 반대매매를 하거나 예탁된 대용증권을 매각할 수 있다.

8.5 미결제약정 수량의 관리

미결제약정이란 옵션거래가 성립된 이후 만기일까지 반대매매, 권리행사, 또는 최종결제 등으로 청산되지 않은 약정을 말한다. 매수 미결제 약정을 보유하고 있는 것을 "매수 포지션(Long Position)을 취하고 있다." 라고 말하고 매도 미결제 약정을 보유하고 있는 것을 "매도 포지션(Short Position)을 취하고 있다." 라고 말한다.

8.6 위탁수수료

증권회사는 투자자의 위탁주문에 대한 매매거래가 성립하거나 최종결제 또는 권리행사에 의한 결제가 발생한 때 투자자로부터 증권회사가 정한 위탁수수료를 징수한다.

구입한 옵션을 만기일까지 보유하지 않고, 도중에 반대 매매하여 손익을 확정짓는 것이 중간 정산이다.

$$\frac{\text{전일비}}{\text{전일종가}} \times 100\% = \frac{2.30}{209.74} \times 100\% = 1.0966\%$$

- 시가 (209.43) : 당일 최초로 형성된 지수
- 고가 (209.99) : 하루 중 가장 높은 지수
- 저가 (205.80) : 하루 중 가장 낮은 지수
- 행사가격 (210.00) : 옵션매입자가 만기일 또는 그 이전에 권리를 행사할 때 적용되는 가격

행사가격(210.00)에 대한 콜옵션 매수가는 9.40이다. 예를 들어, A가 7계약 매수주문을 내고 같은 금액으로 B가 10계약 매도주문을 내게 되면, 주문이 일치된 7계약이 거래로 이루어지고 체결된 7계약이 거래량이 된다.

∗ 이 장의 주요용어

- 파생금융상품
- 유러피언 콜옵션
- 프리미엄
- 옵션만기일
- KOSPI 200
- 사이드카
- 서킷브레이커
- 위탁증거금

연습문제 2

1. 100만원이 주어졌을때 근월물 콜옵션에 전액 투자를 하는 포트폴리오를 작성하시오.

2. 근월물 만기가 지난후 수익률을 계산하시오.

제 3 장

MATLAB 기초

MATLAB(www.mathworks.com)은 미국의 Math Works에서 만들어진 프로그램으로, 1984년도에 소개된 이후로 오늘날 전 세계 50만 이상이 사용하고 있다. MATLAB은 `MATrix+LABoratory`로서 행렬을 기본으로 최적화되어진 프로그램으로 알고리즘 개발, 데이터 수치분석이나 시각화를 위한 컴퓨터 언어이다. C++언어에 비해 사용하기가 편리하다는 장점이 있으나 실행속도가 느리다는 단점을 안고 있다.

특히 MATLAB의 Financial Derivatives Toolbox는 금융 데이터 분석, 모델링, 시뮬레이션 및 최적화를 위한 MATLAB 및 툴박스로 더 자세한 내용은 홈페이지를 참조하기 바란다.

항목	명령어	기능
명령어 나열	,	두 개 이상의 명령어를 한 줄에 표현하려면 콤마(,)를 이용하여 구분

줄넘기기	...	명령어가 너무 길어 다음 줄로 넘어가고 싶을 때 사용 (Command Window에서도 사용 가능)
		(예)
		`plot(x,y,'--rs','LineWidth',2,...`
		`'MarkerEdgeColor','k',...`
		`'MarkerSize',10)`

출력여부	;	Command Window상에서 명령어를 실행하면 화면에 결과가 출력되지만, 세미콜론(;)을 입력하여 실행하면 출력되지 않고 workspace에만 저장된다. (예) `>> a=1;b=2;c=3;`
주석	%	명령어의 앞에 %를 입력하면 주석으로 처리된다.
화면정리	clc	clc 명령어를 입력하고 엔터를 치면 Command Window에 표시되었던 모든 내용들이 지워짐
화면정리	clf	clf 명령어를 입력하고 엔터를 치면 Figure에 나타난 모든 그림이 지워짐
변수삭제	clear	변수 및 배열에 할당된 값들 모두 삭제. (whos로 확인 가능) • 모든 변수를 삭제 : `clear all` • 특정 변수만을 삭제 : `clear 특정변수`

제 2 절 M-file 만들기

MATLAB을 이용하여 원하는 기능을 수행하는 방법은 크게 두 가지로 구분된다.

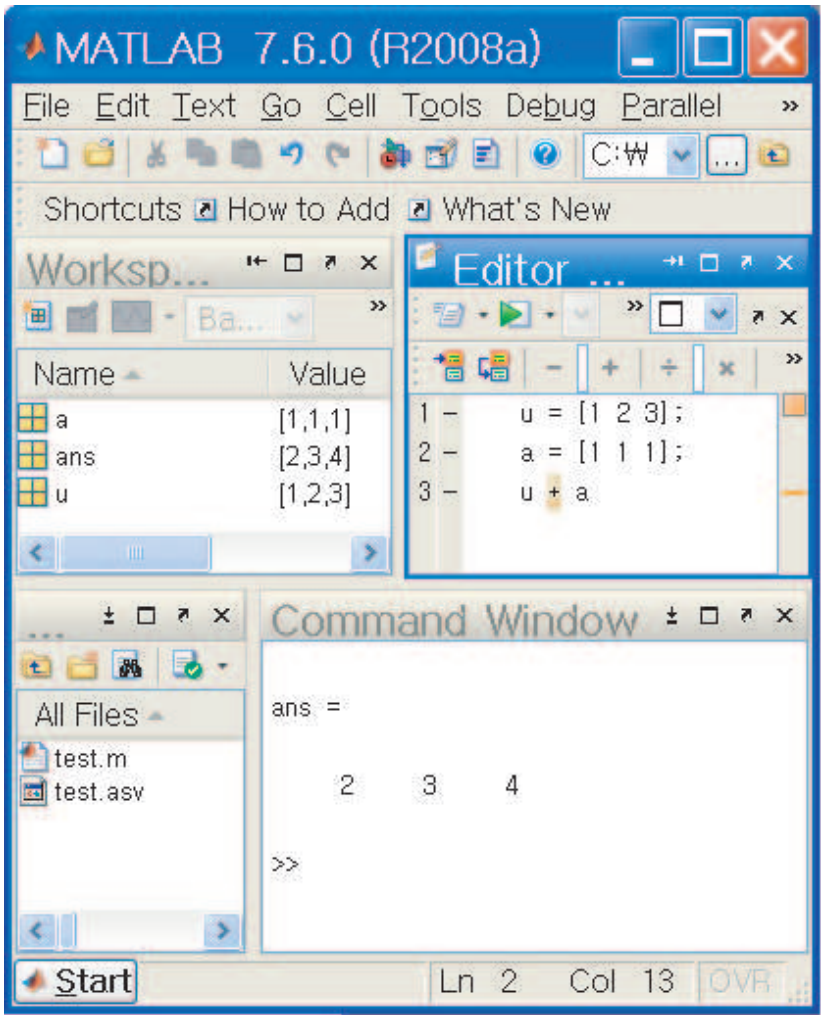


그림 3.1: MATLAB창

첫 번째는 Command Window에 직접 명령어를 입력하는 방법이고, 두 번째는 Script파일을 이용하는 것이다.

MATLAB에서 사용하는 파일을 보통 M-file이라고 부르며 파일의 확장

제 3 절 load 문

파일에 저장된 정보를 읽으려고 할 때 사용한다. 'load'문을 사용하기 위해서는 M-file이 저장된 폴더에 정보가 저장된 파일이 필요하다. 다음문장은 S라는 변수에 파일에 들어 있는 정보를 읽는 문장이다. 다음과 같은 정보가 들어 있는 파일 Data.txt를 읽는 예를 들어 보자.

```
123 234 456
234 345 456
```

[예제] load 문 프로그램

```
S=load('Data.txt')
```

위의 m-file의 실행의 결과로 S는 $2 * 3$ matrix가 되었다.

제 4 절 plot 문

실행결과를 2차원 그래프로 나타내고자 할 때 사용된다. 'plot'문을 사용하기 위해서 2개의 변수가 필요하며 각 변수는 같은 크기의 1차원 배열(벡터)이어야 한다.

- `plot(X,Y)`
  x축은 X, y축은 Y를 값으로 갖는 2차원 그래프를 보여준다.
- `plot(Y)`
  x축의 값을 주지 않으면 default로 x축은 index값을, y축은 Y를 값으로 하는 2차원 그래프를 보여준다
- `plot(X,Y,S))`
  S는 선의 종류, 심볼(symbol) 또는 색을 나타낼 수 있는 옵션값이다.(자세한 내용은 표 4 참고.)

[예제] plot 문 프로그램

```
for i=1:21
    x(i)=0.1*(i-1);
    y(i)=x(i)^2;
end
plot(x,y)
title('Graph of y=x^2')
xlabel('X')
ylabel('Y')
grid
```

[프로그램 실행결과]

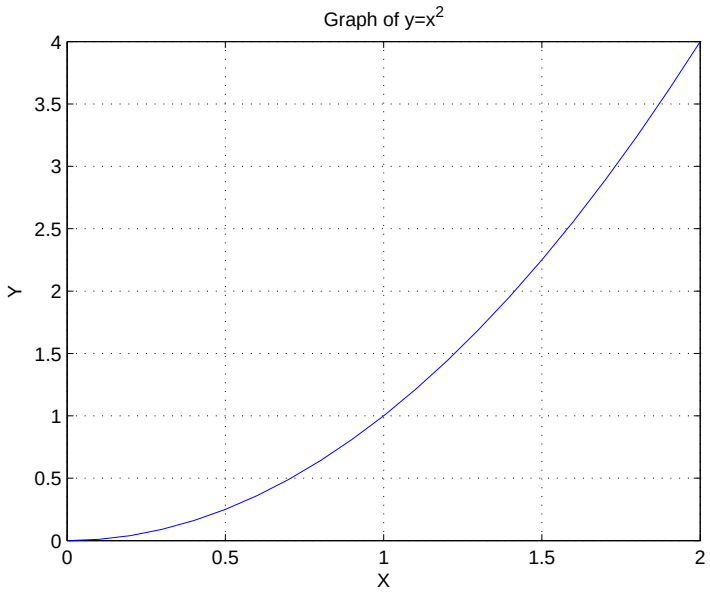


그림 3.4: plot문 프로그램 실행결과

제 4 장

C++ 기초

C++은 C로부터 확장된 언어로써 Bell Lab에서 Bjarne Stroustrup가 C 언어에 몇가지 특징을 추가하여 만들어 졌다. 가장 중요한 추가사항은 classes for object-oriented programming을 지원한다는 점이다. Object-oriented programming은 프로그램의 유지,보수가 쉽다는 큰 장점을 가지고 있다. C++ 언어는 C 언어의 확장된 개념이므로 모든 C 언어의 기능을 지원한다.

1998년 미국 표준 협회(American National Standards Institute, ANSI)는 C++ 언어의 표준규격을 정립하며, 이로써 코드의 이동성을 향상 시켜주었다.(즉, 자신의 compiler를 통해 에러 없이 컴파일을 한 경우 어느 컴퓨터 어느 compiler를 통해서도 에러없이 컴파일이 가능하다.)예를 들어 visual studio 에서 C++로 컴파일에 성공했다면 cygwin, Linux등 C++을 제공하는 모든 compiler에서 사용이 가능하다. 현재의 주요 compiler는 이런 ANSI의 규격을 지원하고 있다. 그렇지만 각각의 compiler는 compiler만의 특성을 지니고 있기 때문에 다른 compiler에서 컴파일을 하기 위해서는 compiler에 맞는 약간의 수정작업이 있어야 한다.

C++ 언어는 C 에 비해 유지, 보수가 편리하며 MATLAB에 비해 연산속도가 빠르고 메모리 할당에서 좀 더 안전하고 간단하게 할당할 수 있는 이점을 가지고 있다. 이 책의 모든 코드는 "C++"로 구현하고 compiler로

제 1 절 Microsoft Visual Studio 2008 소개

Microsoft Visual Studio 2008(이하 VS2008)의 설치를 했다면 '윈도우 시작 버튼'-'프로그램'-'Microsoft Visual Studio 2008'을 선택해서 'VS2008'를 시작한다. 그림 4.1참고

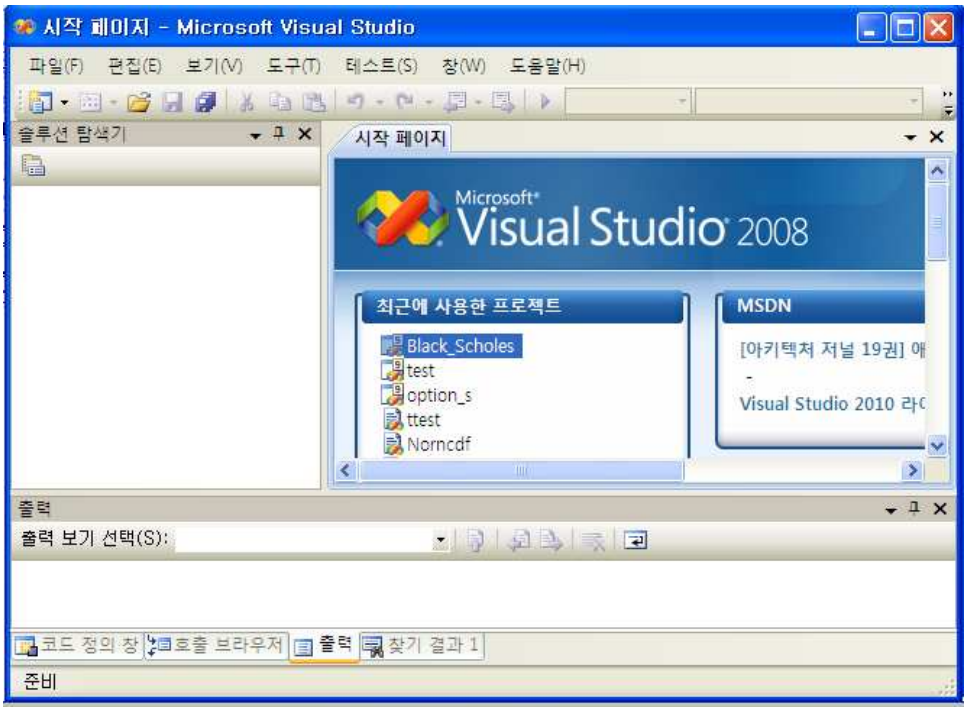


그림 4.1: Microsoft Visual Studio 2008 시작 페이지

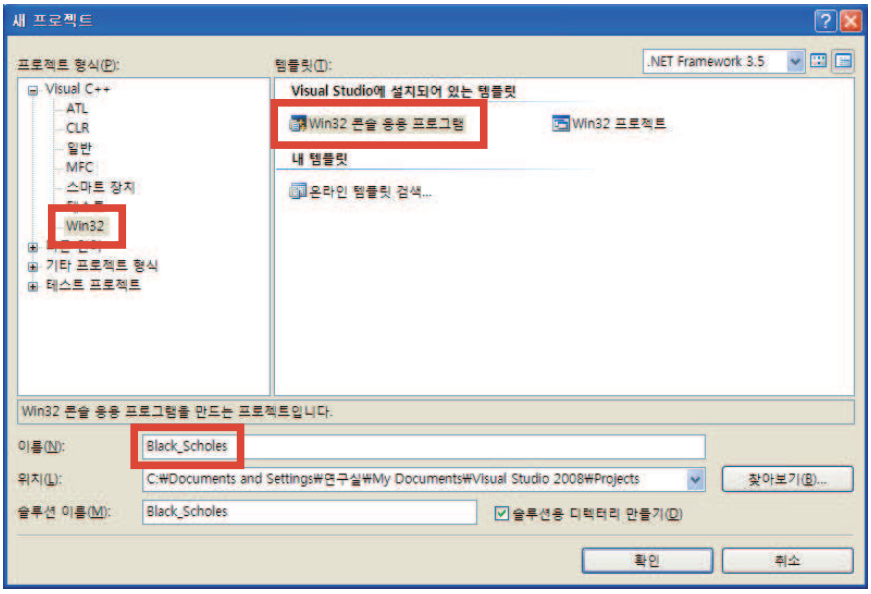


그림 4.3: 프로젝트 생성 2

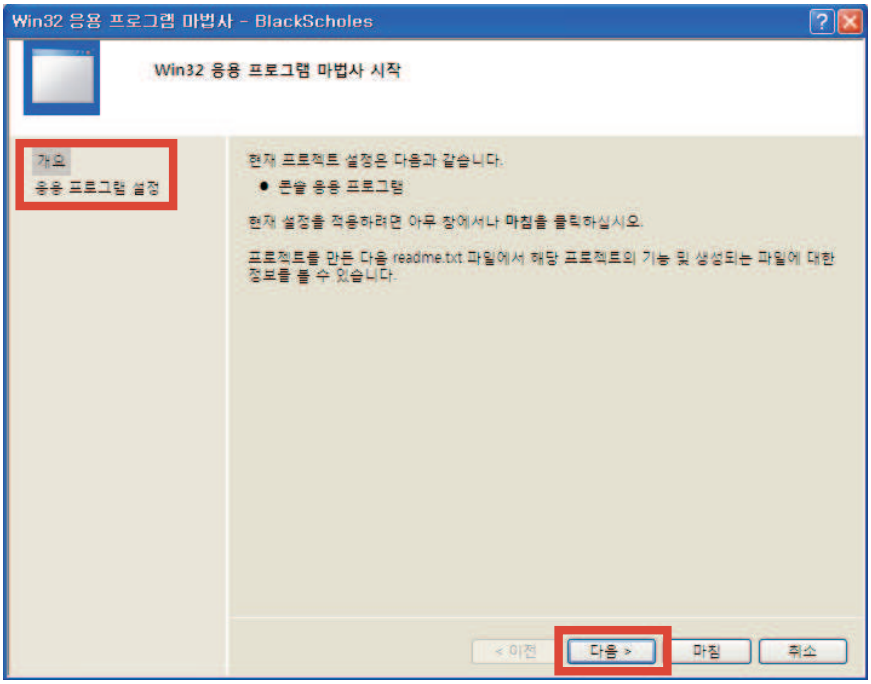


그림 4.4: 프로젝트 생성 3

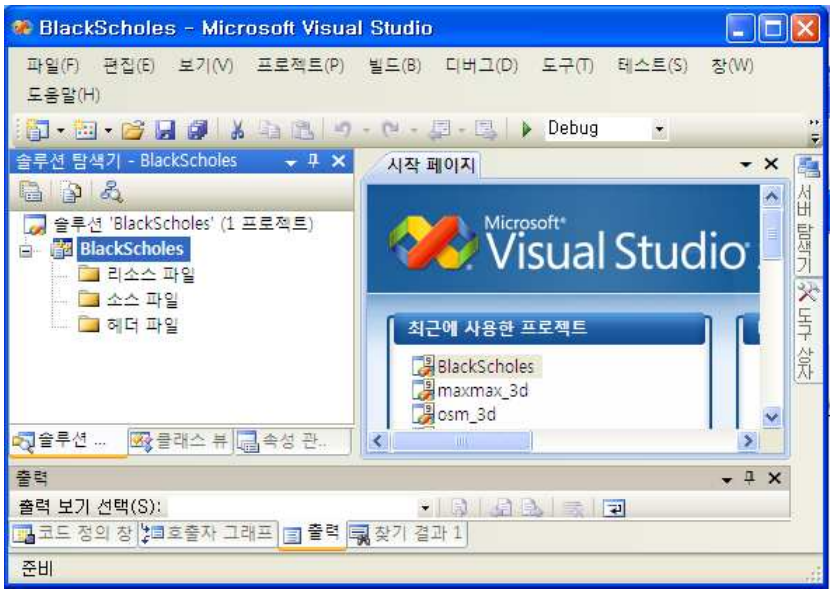


그림 4.6: 프로젝트 생성 5

1.2 C++ 프로그램의 생성

프로젝트를 생성하였다면 이제 C++ 프로그램을 생성할 수 있다.
C++프로그램을 생성하는 방법은
1. 메뉴란의 '프로젝트'-'새 항목 추가' (단축키 : Ctrl+Shift+A) 그림 4.7 참고
2. 새 항목 추가 창이 뜨게 되면 '탬플릿'에서 'C++파일(.cpp)'을 선택하면 C++ 프로그램이 생성된다. 한글이름도 가능하지만 다른 compiler에서는 한글 파일 이름을 인식을 하지 못하는 경우가 있으므로 영어이름으로 해주는 것이 올바르다. 그림 4.8 참고

C++ 프로그램은 main함수에서 시작해서 main함수에서 끝이난다. 즉, main에서 순차적으로 코드를 실행하고 main함수가 끝이 나면 프로그램이 종료하게 된다.
이제 main함수에서 코드를 입력하여 보자. 다음은 "Hello World!!"를 출력하는 코드이다.

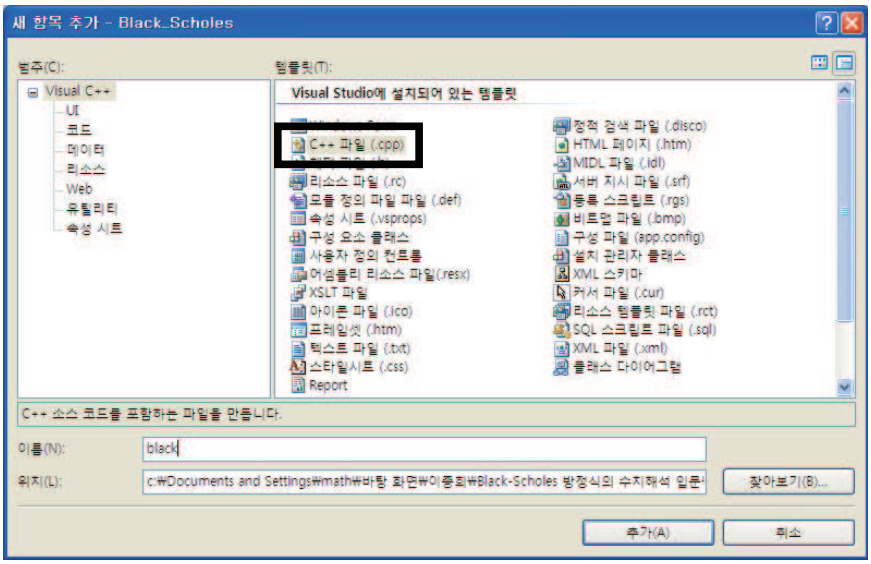


그림 4.8: C++프로그램의 생성 2

상황에 따라서는 프로젝트에 포함된 파일중에서 컴파일에서 제외해야 하는 경우도 생긴다. 이 때는 솔루션 탐색기에서 컴파일을 제외시키려고 하는 파일에 마우스 오른쪽 버튼을 클릭한 후 속성에 들어 가면 다음 그림4.23에서 보이는 대로 컴파일 제외 설정을 한다.

1.4 C++ 프로그램의 실행

프로그램의 실행하기 위해서는 '디버그'-'디버깅 하지 않고 시작' (단축키 : Ctrl+F5)을 클릭한다. 에러없이 실행이 되면 '프로젝트명.exe' 파일이 생성된다. 그림 4.12 참고

실행결과는 그림 4.13 참고

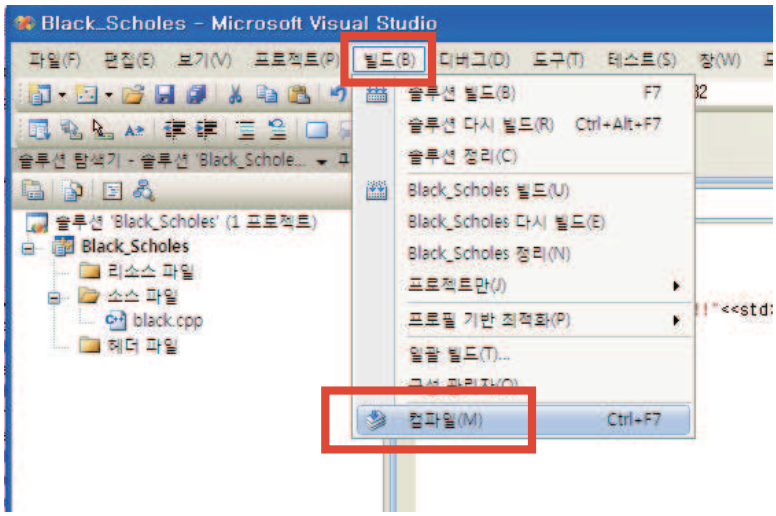


그림 4.10: 컴파일

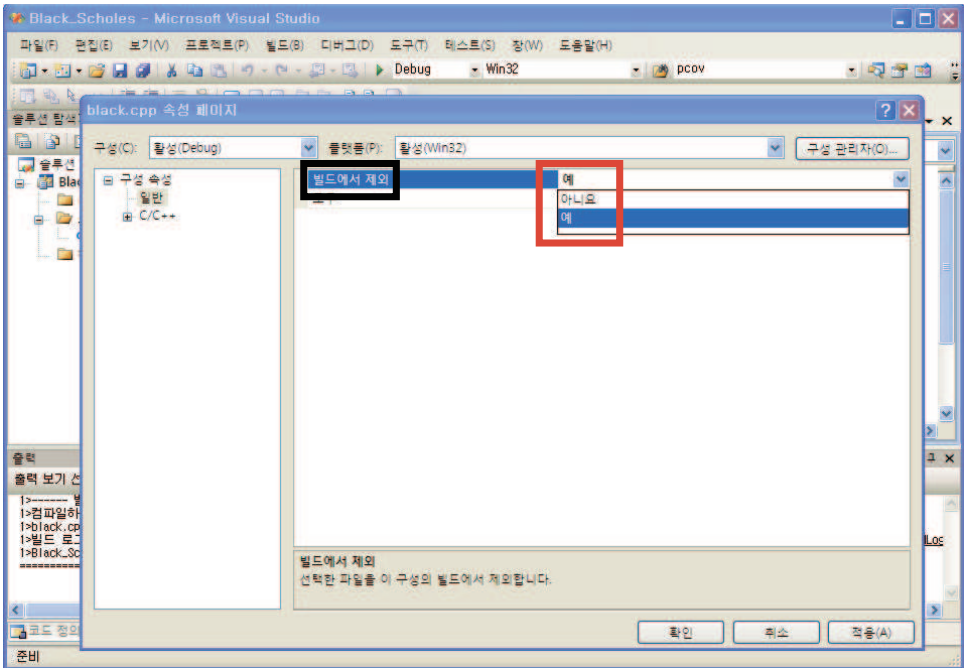


그림 4.11: 컴파일 제외 설정

```
std::cout<<"출력하고 싶은 단어를 입력하세요."<<std::endl;
std::cin>>hello;
std::cout<<"당신이 출력하고 싶은 단어는 "<<hello<<
"입니다."<<std::endl;
return 0;
}
```

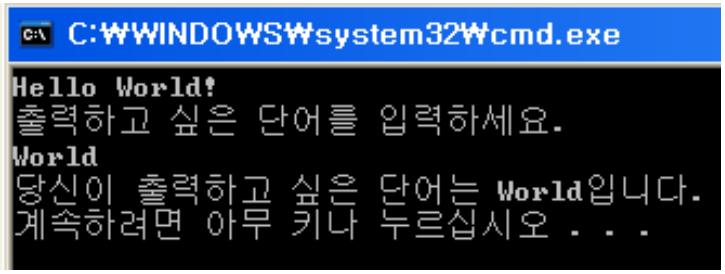


그림 4.14: Hello World 입출력 실행 결과

위 예를 보게 되면 생소한 것들이 보인다. std::cout 과 std::cin 그리고 std::endl 이 그것인데 이들은 std라는 namespace에 정의 되있는 클래스로써 입력과 출력 그리고 행변환을 의미한다.

- std::cout : 출력 클레스
- std::cin : 입력 클레스
- std::endl : 행변환 클레스

이때 매번 std::을 앞에 붙여주기가 너무 많아서 힘든경우 파일 앞부분에

```
using namespace std;
```

라고 선언을 해주면 std::을 생략하고 사용할 수 있다.

2.2.2 파일을 통한 입력과 출력

파일을 통한 입출력을 위해서는 "fstream"이라는 헤더파일을 include해야 한다. 다음은 파일에서의 출력에 관한 예이다. 파일을 불러오는 과정에서 주의해야 할 점이 있다. 파일을 불러오는 과정에서 불러오려는 파일이 존재 하지 않으면 파일을 생성하지만 만약 불러오려는 파일이 존재 한다면 그

```
input.open("scores.txt");
//data 읽기
char Name[80];
char subject[80];
int score;

while(!input.eof())//파일의 끝이 아니면 실행
{
input>>Name>>subject>>score;
cout<<Name<<" "<<subject<<" "<<score<<endl;
}
input.clear();
input.close();//파일 닫기

cout<<"Done"<<endl;

return 0;
}
```

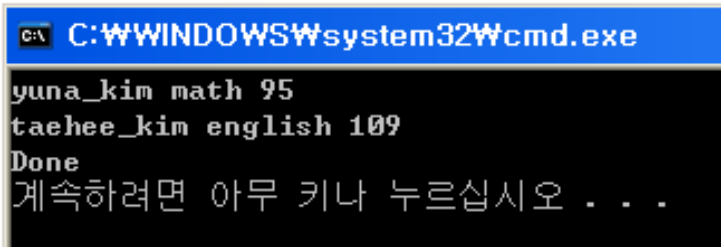


그림 4.15: 파일을 통한 입력결과

2.2.3 파일 열기 모드

이전 장에서는 데이터를 쓰거나 데이터를 읽기 위해서 'ofstream', 'ifstream' class를 이용하였다. 데이터를 쓰고 읽는 일을 동시에 처리 할 수 있는 경우에는 'fstream'을 써서 처리할 수 있다. 만약 'fstream'을 이용 할 경우 사

2.3 변수와 상수

2.3.1 변수

변수라고 하면 변하는 수라고 생각이 되는데 변수는 데이터를 저장할수 있는 메모리 공간에 붙여진 이름(혹은 메모리 공간 자체)을 의미한다. 선언하게 되는 변수에는 다양한 특징을 줄 수가 있는데 크게 정수형 변수와 실수형 변수로 나뉜다. 정수형 변수로는 char형, int형, long형이있고 실수형 변수는 float형, double형 변수가 있다.

자료형(data type)		할당되는 메모리 크기	데이터의 범위
정수형	char	1바이트	$-2^7 \sim 2^7 - 1$
	unsigned char	1바이트	$0 \sim 2^8 - 1$
	int	4바이트	$-2^{31} \sim 2^{31} - 1$
	unsigned int	4바이트	$0 \sim 2^{32} - 1$
	short	2바이트	$-2^{15} \sim 2^{15} - 1$
	unsigned short	2바이트	$0 \sim 2^{16} - 1$
	long	4바이트	$-2^{31} \sim 2^{31} - 1$
	unsigned long	4바이트	$0 \sim 2^{32} - 1$
실수형	float	4바이트	$3,4 \times 10^{-37} \sim 3.4 \times 10^{+38}$
	double	8바이트	$1.7 \times 10^{-307} \sim 1.7 \times 10^{+308}$
	long double	8바이트 혹은 그이상	차이를 많이 보임

변수형 앞에 unsigned 가 붙게 되면 음수부분이 없는 변수가 된다. 음수가 없어지는 만큼 양수로 지정할 수 있다.

```
double d=123456789.101112;
char c='B';
bool b=true;

double ii=static_cast<double> (i);
char iii=static_cast<char> (i);
bool iiii=static_cast<bool> (i);

cout<<"i is"<<setw(6)<<i<<endl;
cout<<"i conversions to"<<" double :"<<setw(6)<<ii<<endl;
cout<<"i conversions to"<<" char :"<<setw(6)<<iii<<endl;
cout<<"i conversions to"<<" bool :"<<setw(6)<<iiii<<endl;

int dd=static_cast<int> (d);
char ddd=static_cast<char> (d);
bool dddd=static_cast<bool> (d);

cout<<"d is"<<setw(16)<<d<<endl;
cout<<"d conversions to"<<" int :"<<setw(6)<<dd<<endl;
cout<<"d conversions to"<<" char :"<<setw(6)<<ddd<<endl;
cout<<"d conversions to"<<" bool :"<<setw(6)<<dddd<<endl;

int cc=static_cast<int> (c);
double ccc=static_cast<double> (c);
bool cccc=static_cast<bool> (c);

cout<<"c is"<<setw(6)<<c<<endl;
cout<<"c conversions to"<<" int :"<<setw(6)<<cc<<endl;
cout<<"c conversions to"<<" double :"<<setw(6)<<ccc<<endl;
cout<<"c conversions to"<<" bool :"<<setw(6)<<cccc<<endl;

int bb=static_cast<int> (b);
double bbb=static_cast<double> (b);
char bbbb=static_cast<char> (b);

cout<<"b is"<<setw(6)<<b<<endl;
```

자료형 변환의 예상되지 않는 변환의 예를 알아보자.

```
#include<iostream>
 using namespace std;

 int main()
 {
 int i=123456;
 double d=123456789.10111213;

 unsigned short us=static_cast<unsigned short>(i);
 short s=static_cast<short>(i);

 float f=static_cast<float>(d);
 short d2f=static_cast<short>(d);

 cout<<fixed;
 cout<<"i is "<<i<<endl;
 cout<<"i conversions to unsigned short is "<<us<<endl;
 cout<<"i conversions to short is "<<s<<endl;
 cout<<endl;
 cout<<"d is "<<d<<endl;
 cout<<"d conversions to float is "<<f<<endl;
 cout<<"i conversions to short is "<<d2f<<endl;

 return 0;
 }
```

2.3.3 Boolean 형 변수

C 언어에서는 참(True)과 거짓(False)를 정수로 나타낸다. 거짓인 경우 '0', 참인 경우 '0이 아닌 정수' 로 표시 했었다. C++ 언어에서는 Boolean 형 변수가 새로이 생겼는데 이는 참과 거짓을 정수로 구분하는 것이 아니라 값이 'True'와 'False' 인 변수이다. 하지만 실제의 저장되는 값에서는 변하지 않았다. 다시 말해 'True=1', 'False=0' 의 값으로 저장된다. 이렇게 함으로써 코드의 가독성이 높아진다.

2.3.4 변수 선언

변수를 선언할 때 시기가 C언어에서는 중요했다. 프로시져상의 모든 변수는 최초 연산이 시작되기 전에 선언이 되있어야 활용이 가능했다. 하지만 C++언어에서는 변수의 선언시기가 중요하지 않게 되었다. 변수는 선언할 변수가 연산에 활용되기 전에 선언하면 된다. 변수를 선언 하는 방법은 선언과 동시에 초기화 하는 방법과 선언후 변수를 초기화는 방법이 있다. 주의해야 할점은 변수를 초기화 하지 않을시 변수는 '쓰레기값'이 들어가게 된다는 것이다. 그렇기 때문에 변수를 선언시 당장 변수를 사용하지 않더라도 '0'으로 초기화를 해주는 것이 프로그램이 안정적으로 구성된다.

```
int val1; //int 형 변수 val1의 선언
val1=10; //변수 선언후 초기화

int val2=20; //int형 변수 val2를 선언과 동시에 초기화
```

변수 선언시 동시 한줄에 선언하는 것 또한 가능하다. 이 때 선언되는 변수 이름과 이름 사이에는 ','(컴마)가 필요 하다.

```
int val1, val2;
/* int형 변수 val1과 val2를
동시에 한줄에 선언 한다.*/
```

2.3.5 변수 선언시 주의 사항

C언어에서 변수는 함수내에서 선언시 변수가 가장 먼저 선언되었지만 C++에서는 변수 선언시 위치는 중요하지 않다. 변수가 필요한 부분에서 편하게 변수를 선언할 수 있다. 그외에 주의 사항이다.

1) 변수 선언시 변수의 이름은 알파벳, 숫자, 언더바(_)로 구성한다.(특수문자는 안된다.)

2) 변수 선언시 알파벳의 대문자와 소문자는 구별된다.

3) 변수의 이름은 숫자로 시작할 수 없고 공백이 포함되서는 안된다.

4) 변수의 이름으로 키워드가 사용되서는 안된다.(키워드란 C++에서 사용되는 명령어와 같은 것이다. 예를 들면 int, return, main,for 등이 있다.)

```
int a=10;
int b=20;
const int* p=&a;
cout<<p<<endl; //포인터p 값
cout<<*p<<endl; //포인터p가 가리키는 값

//*p=30; //error
p=&b; //OK
a=30; //OK
cout<<p<<endl;

return 0;
}
```

다음은 포인터를 const 선언하는 경우이다.

```
#include<iostream>
using namespace std;

int main()
{
//포인터 자체가 상수화
int a=10;
int b=20;
int* const p=&a;
cout<<*p<<endl;
// p=&b; //error 포인터를 바꾸는것은 에러
*p=20; //OK 포인터가 가리키는 값을 변경은 가능
cout<<*p<<endl;
return 0;
}
```

많은 사람들이 const는 에러를 발생시킨다며 const가 선언된 프로그램을 const를 지워버리고 쓰는 경우가 많다. const 키워드는 원래 C언어에는 존재하지 않았지만 C++에서 생긴 개념이다. 하지만 C언어의 표준을 재정립 하면서 C언어에 다시 도입될 정도로 중요한 키워드이다. const는 프

Operator	Name	Description
++var	전치증가연산자	연산이 되기전에 var의 값을 1증가 (선증가, 후연산)
var++	후치증가연산자	연산이 먼저되고 var의 값을 1증가 (선연산, 후증가)
−−var	전치감소연산자	연산이 되기전에 var의 값을 1감소 (선감소, 후연산)
var−−	후치감소연산자	연산이 먼저되고 var의 값을 1감소 (선연산, 후감소)

C++에서 제공하는 관계 연산자 들을 알아 보자. 이를 배우기 앞서 자료형 bool 에 대해 알아보자. C 언어에서는 참과 거짓을 나타내기 위해 정수를 사용하여 거짓은 0로 표현하고 참은 0이 아닌 수로 표현했다. (대표적으로는 1을 사용) 하지만 C++에서는 boolean이라는 자료형으로 true 와 false를 사용한다. 즉, 참은 true 이고 거짓은 false 이다.

Operator	Example	Result
<	1 < 2	true
>	1 > 2	false
<=	1 <= 2	true
>=	1 >= 2	false
==	1 == 2	false
! =	1! = 2	true

C++에서 제공하는 로직 연산자들을 알아 보자.

```
const int a=4;
double math[a];
```

```
#include<iostream>
using namespace std;

int main()
{

// dataType arrayName[index]   array 선언 구조
double myList[3];
myList[0]=0.0;
myList[1]=1.0;
myList[2]=2.0;
cout<<"myList[2] is "<<myList[2]<<endl;
/* myList라는 int형 3개 0,1,2 가
들어간 array가 만들어 졌다*/


/*array선언과 초기화를 동시에 해보자
뒤따르는 선언도 같은 결과를 가져온다.*/

double yourList[3]={0.0,1.0,2.0};
cout<<"yourList[2] is "<<yourList[2]<<endl;

//2차원 array는 다음과 같이 선언한다.
double hisList[3][2]={{1,2},{2,3},{3,4}};
cout<<"hisList[1][1] is "<<hisList[1][1]<<endl;

return 0;
}
```

2.6 반복문과 분기문(Loop and Select statements)

C++프로그램을 만들다 보면 프로그램 상에서 양갈래, 혹은 그 이상의 갈래로 나누어져 연산을 하는 경우나 똑같은 연산을 반복해서 해야 경우가 생

```
 //1/count가 tol 보다 큰경우 반복, 그렇지 않은 경우 반복하지 않음
{
cout<<"count"<<" "<<count<<endl;
count=count+1;
if (count >100)
break;
}
return 0;
}
```

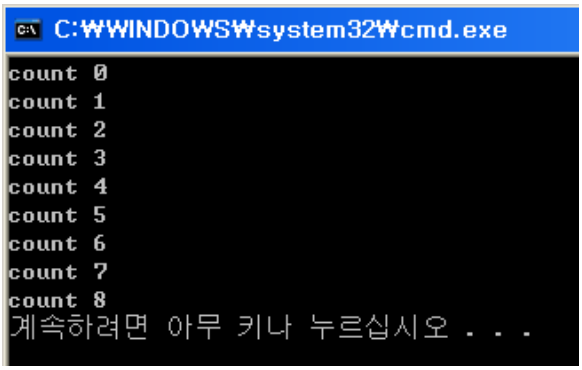


그림 4.20: While 문 실행 결과

while 문과 비슷하게 do-while문이 있다. 이는 while문과 거의 비슷하다. while문과 다른점은 boolean 표현이 문장의 끝에 있어서 적어도 한번은 프로세스가 진행 되고 loop조건을 확인한 후 loop를 실행할지 결정을 하는 문장이다.

```
#include<iostream>
using namespace std;

int main()
{
int count=0;
double tol=0.12;
```

```
int main()
{
int num;
cout<<"9999이하의 숫자를 입력하세요"<<endl;
cin>>num;
if (num>=1000) //1000을 넘는 수를 입력했을때 실행
{
cout<<"네자리수 입니다."<<endl;
}
else if (num>=100) //1000을 넘지는 않지만 100을 넘을때 실행
{
cout<<"세자리수 입니다."<<endl;
}
else if (num>=10) //100을 넘지는 않지만 10을 넘을때 실행
{
cout<<"두자리수 입니다."<<endl;
}
else //위의 세가지 경우를 제외한 모든경우에 실행
{
cout<<"한자리수 입니다."<<endl;
}
return 0;
}
```

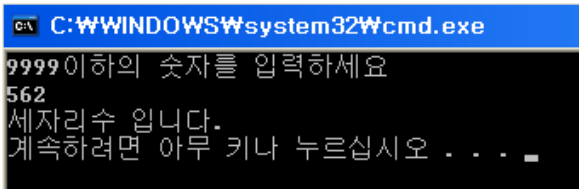


그림 4.21: if문의 실행 결과

switch 문은 사실 if 문으로 처리하는 것이 가능하다. 하지만 선택해야 하는 경우의 수가 많다면 true 또는 false로 경로를 선택하는 if문의 경우 프로그램을 쓰기도 힘이 들지만 직관적으로 읽어내기에도 불편이 따르게 마련이다. 또한 if문은 모든 조건문을 다 검사하기 때문에 경우의 수가 많을

```
{
double P;
switch(nType)
{
case 0: P = A*pow(1+r,n);
break;
case 1: P = A*(1+n*r);
break;
}
return P;
}
```

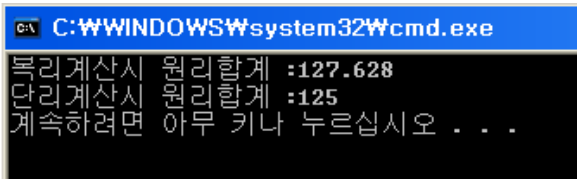


그림 4.22: if문의 실행 결과

2.7 함수

C++에서 가장 기본적인 단위는 함수이다. 함수라면 알다시피 인자를 함수에 대입하여 결과를 이끌어내는 형태를 함수라고 한다. C++ 언어에서는 다음과 같은 형식으로 함수를 선언, 정의 하게 된다.

반환형 함수이름 (변수형 인자){함수의 몸체}

함수의 선언, 정의하게 될때 만약 함수의 몸체 부분이 1줄만 존재한다면 중괄호는 생략가능하다. 간단하게 $f(x) = 2 + x$ 를 선언해 보자.

```
int func (const int x) //정수형 변수 x을 인자로 갖는 func함수 선언
{
```

```
return 0;
}
void call_by_value(int a, int b)
{
int temp;
temp=a;
a=b;
b=temp;
}
void call_by_reference(int &a, int &b)
{
int temp;
temp=a;
a=b;
b=temp;
}
```

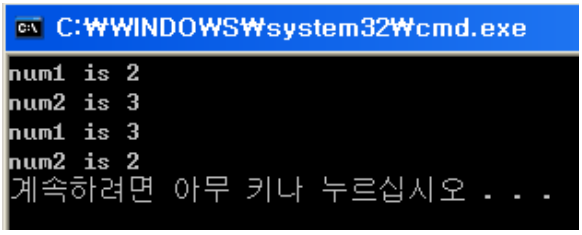


그림 4.23: call by value 와 call by reference 의 차이

2.7.2 함수의 Overloading

함수의 overloading 이란 같은 특징의 함수들을 하나의 이름으로 사용하는 것이다. C언어에서는 max라면 int형 변수를 두개 입력하고 그 중 큰 수를 출력하는 함수이다. 하지만 만약 두 double형변수의 maximum값을 찾으려면 어떻게 될까? 다른이름의 함수를 만들어 내는 수 밖엔 없었다. 하지만 C++에서는 해법이 있다. 그것이 함수의 Overloading 이다. 다시 말하면 이름은 같지만 인수들의 형태, 갯수만 바꾼다면 C++ 에서는 다른 함수처럼 쓸 수 있다. 다음 예를 통해 알아보자.

```
return 0;
}
```

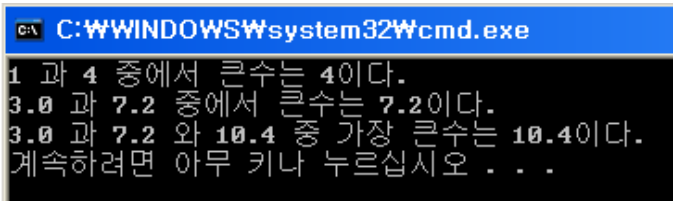


그림 4.24: 함수의 overloading

함수의 Overloading 을 사용시 주의 할점은 함수는 인수의 개수나 인수의 자료형에는 Overloading을 하지만 반환형이 다르다고 Overloading 되지는 않는다. 이는 코드상에서 보면 알겠지만 함수를 선언 후 함수를 사용하게 되면 반환형은 따로 쓰지 않기 때문에 compiler는 어떤 함수를 호출하는지 애매하다고 판단하고 컴파일 에러를 발생시킨다.

2.7.3 Default 인자

Default란 기본적으로 선택되는 사항이라는 의미이다. 카메라를 예를 들면 사진을 찍을 때 플래쉬는 터뜨리는지 셔터속도는 얼마로 하는지 사진을 연속으로 몇장을 찍을지 선택하지 않더라도 카메라를 켜자마자 셔터를 누르면 사진이 찍히게 되어있다. 이러한 기본적으로 선택되어 있는 것을 Default라고 설명할 수 있다. 카메라에 기본적으로 선택되어 있는 값을 Default인자라고 할 수 있다. 다음의 예를 통해 알아보자.

```
#include<iostream>
using namespace std;

int price(int num_shirts=1, int num_pants=1, int num_shoes=1)
//기본적으로 셔츠1벌, 바지1벌, 신발 1켤레를 사는것을 Default로 정의
했다.
{
int shirts_price=10000;//셔츠 1벌의 가격
```

2.8 Class

객체 지향형 프로그램(Object-oriented programming) 은 객체(object)를 포함한 모든 프로그래밍을 의미하는 것으로 생각할 수 있다. 객체(object)란 실생활에서 보게 되는 사람, 물건 등과 비슷한 의미 이다. 하나의 프로그램을 구성할때 여럿의 클래스들이 모여서 구성하게 되는데 이를 로봇에 비유하면 머리를 담당하는 로봇과 몸체를 담당하는 로봇, 그리고 팔,다리를 담당하는 로봇들이 모여 하나의 완성된 형태의 로봇이 나오게 되는 데 이것과 흡사하다. C++ 에서는 이런 부분적인 로봇들이 클래스 객체 인것 이다. 이러한 객체(object)들은 자기만의 이름, 상태, 수행능력을 가지고 있다. 추상화를 거쳐서 생성된 속성을 클래스 정의에서 멤버변수라 한다. 즉, 클래스내에서 속성을 정의하기 위해 선언하는 변수이다. 예를 들어 로봇팔에 대한 '제품번호', '제품이름' 등으로 생각할 수 있다. 수행능력이란 클래스가 가지고 있는 function을 의미한다. object의 특정한 행동을 처리하는 함수의 형태를 멤버함수라 한다. 로봇팔은 '움직인다', '공격하다' 등으로 생각할 수 있다.이를 멤버 함수라 한다.이렇게 객체 지향형 프로그램을 구성하게 되면 차후의 유지 및 보수에서 보다 적은 비용과 시간과 인력으로 해결할 수 있다. 절차지향형의 프로그램의 경우 프로그램의 일부를 바꾸는 것은 사실 프로그램에 걸친 모든 부분을 수정해야하는 문제를 안게 되지만 객체지향의 경우는 마치 로봇에서 다리가 고장나면 다리만 떼어내서 수리하고 다시 결합시키면 작동하는 것처럼 프로그램에서 일부를 수정하는 것은 수정이 필요한 부분의 클래스만을 수정하면 된다. 프로그램에 전체적인 틀은 바꾸지 않아도 되고 클래스 별로 분업화가 가능하기 때문에 효율성과 유지, 보수에 비용이 적게 드는 셈인것이다.

2.8.1 Class의 정의

class는 크게 "class 이름", "data field", "method" 이 세가지로 구성되어 있다고 할 수 있다.

- class 이름 : class의 이름

- data field : object의 속석을 나타내는 field

으로는 class 이름과 같고 반환값이 없으며 overloading이 가능하다. 그리고 반드시 public 이어야 한다. 생성자는 그 쓰임새로 인해 default 생성자, 인자가 있는 생성자, 복사 생성자로 나눌수 있다.
소멸자란 class가 소멸될 때 호출 되는 함수 이다. 정적할당의 경우 class가 소멸할 때 자동적으로 해제가 되지만 동적할당은 그렇지 않기 때문에 보통 소멸자에서 동적할당한 멤버변수를 해제한다. 특징으로는 class이름 앞에 ‘ ’이 붙고 반환값이 없으며 overloading 불가능하다. 생성자와 마찬가지로 public 이어야 한다.

- default 생성자 : 인자가 생략된 생성자
- 인자가 있는 생성자 : 인자를 이용하여 초기화 하는 생성자
- 복사 생성자 : 인자가 class인 생성자, 인자class의 멤버 변수를 복사한다.

```
#include<iostream>
#include<cmath>
using namespace std;

class Equilateral
{
public:
double m_edge;
Equilateral(){};//default 생성자
Equilateral(double nedge)//인자가 있는 생성자
{
cout<<"인자가 있는 생성자가 실행 되었습니다."<<endl;
m_edge=nedge;
}
Equilateral(const Equilateral &Eq)//복사 생성자
{
cout<<"복사 생성자가 실행 되었습니다."<<endl;
m_edge=Eq.m_edge;
}
```

접근 제어 지시자	의미
public	클래스 내부와 외부에서 접근 허용
protected	상속관계에서 상속받은 클래스와 상속하는 클래스의 접근 허용
private	해당 클래스 내에서만 접근 허용

외부에서 private나 protected에 접근이 불가능 하지만 class 멤버함수를 이용하면 private나 protected에 접근이 가능하다. 정해진 사항은 없지만 많은 경우 Set, Get이라는 이름으로 많은 사람들이 사용한다. Set, Get은 정해진 함수라기 보다는 멤버함수를 통하여 외부에서 멤버변수로 접근하려고 하는 예이다. 다음 예를 통해 사용하여 보자.

```
#include<iostream>
#include<cmath>
using namespace std;

class Equilateral
{
public:
double m_edge;
Equilateral(){};
Equilateral(double nedge)
{
cout<<"인자가 있는 생성자가 실행 되었습니다."<<endl;
m_edge=nedge;
}
Equilateral(const Equilateral &Eq)
{
cout<<"복사 생성자가 실행 되었습니다."<<endl;
m_edge=Eq.m_edge;
}
double area()
{
return m_edge*m_edge*sqrt(3.0)/4;
}
```

```
~Shape(){};
void print(void);
void Set_area(double narea){m_area=narea;}
double Get_area(void){return m_area;}
protected:
double m_area;
};
void Shape::print(void)
{
cout<<"도형의 넓이는 "<<m_area<<endl;
}

class Equilateral :public Shape
//Equilateral class는 Shape class 를 상속하고 있다.
{
public:
double m_edge;
Equilateral(){m_area=m_edge*m_edge*sqrt(3.0)/4;}
/*m_area는 Equilateral class의 멤버변수가 아니지만
Shape class의 멤버변수 이므로 상속관계에서 print 함수를 가져다 사용할 수
있다.*/
Equilateral(double nedge)
{
m_edge=nedge;
m_area=m_edge*m_edge*sqrt(3.0)/4;
}
Equilateral(const Equilateral &Eq)
{
m_edge=Eq.m_edge;
}
void Set_edge(double nedge){m_edge=nedge;}
double Get_edge(void){return m_edge;}
};

int main()
{
```

pInt[0]에 할당된 메모리란 해제를 하게 되고 나머지 9개의 메모리가 남게 되어 메모리 누수를 발생하게 된다. 그렇게 되면 알 수 없는 오류를 발생시킬 수 있고 많은 누수가 일어날 경우 프로그램이 느려지는 현상이 나타날 수 있다.. 동적 할당을 할 경우 compiler가 자동적으로 해제를 해주지 않기 때문에 반드시 명시적으로 해제를 해주어야 한다.

2.10 열거형(enum)

열거형은 상수를 정의를 하는데 사용을 한다. 어떤 변수가 가질 수 있는 특징이 일정 범위에로 정의로 정해져 있다면 ♯define으로 정의를 할 수 있겠지만 enum을 쓰는 것이 더 바람직하다. 이름을 명시를 해줄 때는 대문자를 쓰는 것이 일반적이다 열거형으로 나타내는 방법은

```
enum 변수 이름 {이름1, 이름2, 이름3…}
```

열거형은 내부적으로 정수로 처리되며 기본적으로는 0부터 1씩 증가를 한다. 만약 값을 다르게 정의를 하고 싶다면 이름뒤에 '='를 붙이고 정의를 하면 된다. 예를 들어

```
enum enumMark { EAST , WEST,  SOUTH,  NORTH};
 또는
enum enumMark { EAST =100, WEST,  SOUTH,  NORTH};
```

이와 같이 열거형을 쓰게 되면 기억하기 쉽고 소스의 가독성이 높아질 뿐만 아니라 에러발생률도 낮출 수 있다.

함수	반환형	설명
pushback(value)	void	vector에 value를 마지막 값으로 추가
popback()	void	vector의 마지막 값을 제거
size()	unsigned int	vector의 크기
at(index:자료형)	자료형	vector의 index번째 원소를 반환
empty()	bool	vector이 비어있는 경우
clear()	void	vector의 내용을 제거 한다
swap(v1 : vector)	void	벡터의 내용을 바꾼다.

vector 선언과 할당은 다음과 같다.

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
//1차원 벡터 선언(정적 할당)은 다음과 같다.
//vector<자료형> 벡터이름(벡터크기);
vector<double> myList(3);
/*myList 라는 double형을 3개 가지는 벡터가
생성되었고 벡터의 원소는 모두 "0"으로 초기화
된다. 자료를 넣는 방법은 array와 동일하다.*/
myList[0]=0;
myList[1]=1;
myList[2]=2;

//myList 출력
for(unsigned int i=0; i<myList.size();i++)
cout<<"myList["<<i<<"]"<<" is "<<myList[i]<<endl;
cout<<endl;
//1차원 벡터 선언(동적 할당)은 다음과 같다.
```

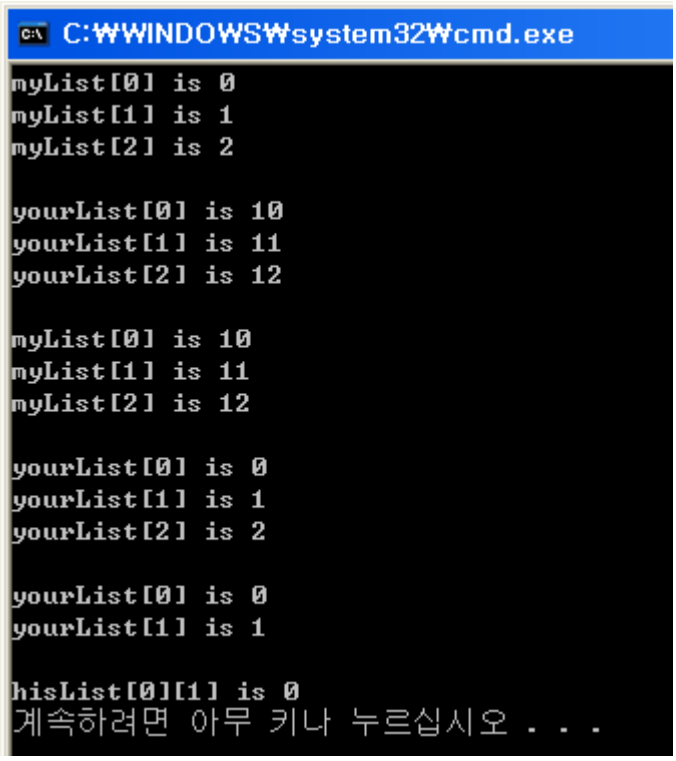


그림 4.29: vector 선언 실행 결과

3.2 cmath

C++ 언어에서 수학적인 함수가 정의 되어 있는 STL은 cmath이다. cmath를 include하면 여러가지 수학적 함수들을 사용할 수 있다.

크게 세가지로 분류 하면 삼각함수, 지수,로그함수, 그외 함수들로 나눌수 있는데 삼각함수 먼저 살펴 보자.

제 5 장

이또의 보조정리(Itô Lemma)

금융문제와 같이 시간과 연동된 변동에 대한 수학적 모델은 대개 브라운 운동과 같은 확률과정을 사용하여 나타낸다. 브라운 운동의 엄밀한 수학적 기초를 닦은 미국의 수학자 Norbert Wiener의 이름을 따서 브라운 운동을 위너과정(Wiener Process)이라고도 한다. 금융시장에서 기초자산의 가격경로는 위너과정을 따른다고 가정하는 경우가 많다. 이 절에서는 위너과정에 대해 자세히 알아보기로 한다.

제 1 절 정규 분포

정규분포(normal distribution) 를 따르는 확률변수 $X \sim N(\mu, \sigma^2)$의 확률밀도함수(probability density function) $f(x)$는 다음과 같다.

$$P(X = x) = f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

여기서 μ는 평균 그리고 σ는 표준편차이다. 확률변수 X의 기대값을 다음과 같이 정의하자.

$$\mathcal{E}[X] = \int_{-\infty}^{\infty} x f(x) dx = \int_{-\infty}^{\infty} \frac{x}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx. \tag{5.1}$$

극좌표로 변형시키기 위하여, $x = r\cos\theta,\ y = r\sin\theta$라고 두자. 그러면,

$$\begin{aligned} I^2 &= \int_0^{2\pi}\int_0^{\infty} \frac{1}{2\pi\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right) r\,dr\,d\theta \\ &= \int_0^{2\pi}\left[-\frac{1}{2\pi}\exp\left(-\frac{r^2}{2\sigma^2}\right)\right]_0^{\infty} d\theta = \int_0^{2\pi}\frac{1}{2\pi}d\theta = \left[\frac{1}{2\pi}\theta\right]_0^{2\pi} = 1. \end{aligned}$$

I가 양수이므로, $I = 1$이다. 다음 식이 성립한다.

$$\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-(\sigma^2 t+\mu))^2}{2\sigma^2}\right)dx = 1.$$

따라서, 다음 식이 성립한다.

$$M_X(t) = \exp\left(\mu t + \frac{\sigma^2}{2}t^2\right).$$

이 식을 미분하면, 다음 식들이 성립함을 알 수 있다.

$$\begin{aligned} \frac{dM_X(t)}{dt} &= (\mu+\sigma^2 t)\exp\left(\mu t+\frac{\sigma^2}{2}t^2\right), \\ \frac{d^2M_X(t)}{dt^2} &= \sigma^2\exp\left(\mu t+\frac{\sigma^2}{2}t^2\right) + (\mu+\sigma^2 t)^2\exp\left(\mu t+\frac{\sigma^2}{2}t^2\right). \end{aligned}$$

따라서, 다음 식들이 성립한다.

$$\begin{aligned} \mathcal{E}[X] &= M_X{}'(0) = \mu, \\ \mathcal{E}[X^2] &= M_X^{(2)}(0) = \sigma^2+\mu^2, \\ Var[X] &= \mathcal{E}[X^2] - \mathcal{E}[X]^2 = \sigma^2+\mu^2-\mu^2 = \sigma^2. \end{aligned}$$

□

정리

X가 확률밀도함수 $f(x)$를 갖는 연속확률변수이면, 임의의 실수값 함수 g에 대해 다음이 성립된다.

$$\mathcal{E}[g(X)] = \int_{\infty}^{\infty} g(x)f(x)dx$$

σ가 1이면, 이 확률과정을 표준 Brown운동이라 한다.

일반화 Brown 운동

만일 확률변수 $S(t)$가 위의 Brown 운동의 첫번째, 두번째, 그리고 세번째 조건을 만족하고 평균이 μt이고 분산이 $\sigma^2 t$인 정규분포를 따르면 일반화 Brown 운동이라 부른다. 이 μ를 추세모수(drift parameter)라 한다.

일반화된 Brown 운동에서 파생된 기하 Brown 운동은 다음과 같이 정의된다.

기하 Brown 운동(Geometric Brownian Motion, GBM)

일반화된 Brown 운동 $\{S(t)|t \leq 0\}$에 대해서, 식 $Z(t) = e^{S(t)}$로 정의된 $\{Z(t)|t \leq 0\}$를 기하 Brown운동이라 한다.

제 3 절 자산 가격을 위한 간단한 모델

효율적 시장가설(effective market hypothesis)은 시장에서 기본적 정보와 기술적 정보를 포함한 과거의 모든 정보가 금융자산의 가격에 반영되어 있다고 가정한다. 먼저, 자산 가격에서 절대적인 변화는 그 자체로서 유용한 정보는 아니다. 예를 들어 1포인트의 변화는 자산 가격이 200p보다 20p일 때 좀 더 의미 있다는 것이다. 따라서 변화의 이런 상대적 측정은 어떠한 절대적 수치보다 그것의 척도를 좀 더 명확하게 나타낼 수 있다.

시점이 t일 때 기초자산 가격을 S라고 가정하자. 그림 5.1처럼 S가 $S + dS$로 변하는 작은 시간구간 dt를 생각해보자. 그렇다면 dS/S인 상대적 변화를 어떻게 모형화할 수 있을까? 가장 일반적인 접근법은 dS/S를 μdt와 σdX의 두 부분으로 나누는 것이다.

μdt는 무위험채권에 투자한 금액의 수익처럼 예측가능하고 확정적인 것을 나타낸다. μ는 단위시간당 기대수익금(drift)로써 dt시간동안 자산가격의 평균성장률 측정치를 의미한다.

dS/S에 대한 두 번째 부분 σdX는 예상치 못한 소식과 같은 외부 요인에 의한 자산 가격의 랜덤한 변동을 모형화한다. 그것은 평균이 0인 정규

작위성(randomness)을 포함하는 dX는 위너과정(Wiener process)을 따르고 다음의 성질을 갖는다;

- dX는 정규분포로부터 나온 확률 변수이다.
- dX의 평균은 0이다.
- dX의 분산은 dt이다.

ϕ가 평균 0 분산 1의 표준정규분포 $N(0,1)$을 따른다면 dX는 정규분포 $N(0,dt)$를 따르므로 $dX = \sqrt{dt}\phi$로 표현된다. $X(t)$는 미분가능이 아니므로 dX을 정의할 수는 없지만 $dt \to 0$으로 했을 때

$$dX = \sqrt{dt}$$

로 된다.

제 4 절 이또의 보조정리(Itô Lemma)

시계열 $S(t)$의 변화량 dS가 다음의 식에 따라 움직이고 있다고 하자.

$$dS = a(S,t)dt + b(S,t)dX.$$

일반화된 위너과정의 상수 a와 b를 S와 t에 대한 함수 $a(S,t), b(S,t)$로 일반화한 것을 이또과정이라고 하며, 따라서, 이 시계열 $S(t)$의 움직임을 이또과정이라 할 수 있다. 이또과정을 이용하면 이또의 보조정리(Itô lemma)를 이끌 수 있다.

이또의 보조정리

S가 이또과정

$$dS = a(S,t)dt + b(S,t)dX$$

를 따를 때, S와 t의 함수 $V(S,t)$의 동향은

$$dV = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}b^2(S,t) + \frac{\partial V}{\partial S}a(S,t)\right)dt + \frac{\partial V}{\partial S}b(S,t)dX$$

를 따른다.

이 되고, 이를 식 (5.7)에 대입하면

$$dV = \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma dX = \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma\sqrt{dt}\phi$$

를 얻게 된다. 이 식은 상수 계수를 갖는 확률 미분방정식이고, dV는 정규분포를 만족한다. $t = ndt$라 하자. 그러면,

$$\begin{aligned} dV_1 &= \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma\sqrt{dt}\phi_1 \\ dV_2 &= \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma\sqrt{dt}\phi_2 \\ &\vdots \\ dV_n &= \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma\sqrt{dt}\phi_n \end{aligned}$$

위 식들의 양변을 다 더하면,

$$V = V_0 + \left(\mu - \frac{\sigma^2}{2}\right) t + \sigma\sqrt{t}\phi, \tag{5.8}$$

여기서, $\phi_1 + \phi_2 + \cdots + \phi_n = \sqrt{n}\phi$ 관계를 사용하였다 (식 (10.14) 참고).

$V(S,t)$의 pdf는 $-\infty < V < \infty$에 대해

$$\frac{1}{\sigma\sqrt{2\pi t}} e^{-\frac{(V - V_0 - (\mu - \frac{\sigma^2}{2})t)^2}{2\sigma^2 t}}$$

이다. 이처럼 확률변수 S에 $V = \log(S)$를 취한 결과가 정규분포를 따를때, S를 대수정규분포(log-normal distribution)를 따른다고 한다.

제 6 장

옵션가격이론

제 1 절 옵션 가격 결정 모형의 Black-Scholes 편미분 방정식

Black-Scholes 편미분 방정식(partial differential equation:PDE) [1] 은 미국의 Fisher Black 교수와 Myron Scholes 교수에 의해 개발된 옵션 가격 결정 모형으로서 옵션 이론 가격을 산출할 때 일부 수정된 모델이 현재 널리 이용되고 있다. 이 모델을 이용하면 기초자산가격(S), 행사가격(E), 잔존기간(T), 무위험이자율(r), 기초자산가격의 변동성(σ)의 값들로 콜옵션과 풋옵션의 이론가격을 직접 계산할 수 있다.

Fisher Black과 Myron Scholes는 확률미적분학을 사용하여 유러피언 옵션에 대한 공정가격을 결정하기 위한 편미분 방정식을 유도했다. 이러한 Black-Scholes 이론은 기본적으로 주식(위험자산)과 채권(무위험자산)으로 이뤄진 포트폴리오의 만기 가치가 '주식시장이 어떤 상황을 거치더라도' 정확히 옵션의 만기 payoff와 일치하도록 하는 트레이딩 전략을 구체적으로 제시하여 준다. 1970년 Black-Scholes 공식의 개발과 함께 1973년 개설된 시카고 옵션거래시장은 세계의 옵션시장을 크게 발전시키는 역사적 계기가 되었다.

옵션 이론가격를 구하기 위해서는 다음의 다섯 가지 변수를 반드시 알아야 한다.

이다. 여기서, $\frac{\partial V}{\partial S}$는 상수로 간주한다. 이 식의 dV와 dS에 각각 이또 렘마(6.2)와 이또과정 (6.1)를 대입하면

$$\begin{aligned} d\Pi &= dV - \frac{\partial V}{\partial S}dS \\ &= \left(\frac{\partial V}{\partial S}\mu S + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)dt - \frac{\partial V}{\partial S}\sigma S dX \\ &\quad -\frac{\partial V}{\partial S}\left(\mu S dt + \sigma S dX\right) \\ &= \left(\frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\right)dt. \end{aligned} \tag{6.4}$$

한편, 무위험 이자율을 r로 하면 dt 시간동안 포트폴리오의 변화량은

$$d\Pi = r\left(V(S,t) - \frac{\partial V}{\partial S}S\right)dt \tag{6.5}$$

가 성립되는 셈이다. 따라서, 두식 (6.4)과 (6.5)로 부터

Black-Scholes 미분방정식

$$\frac{\partial V(S,t)}{\partial t} = rV(S,t) - rS\frac{\partial V(S,t)}{\partial S} - \frac{\sigma^2 S^2}{2}\frac{\partial^2 V(S,t)}{\partial S^2} \tag{6.6}$$

을 도출하게 된다. 여기에 옵션의 만기시점에서 지불되는 지불금액함수를 경계조건으로 하면 다음의 식(6.7)을 만족하며, 그림 6.1를 따르게 된다.

$$V(S,T) = \begin{cases} S - E & \text{if } S \geq E \\ 0 & \text{if } S < E \end{cases} \tag{6.7}$$

식 (6.6)은 식(6.7)의 만기조건을 가지는 Black-Scholes의 편미분 방정식이다.

1.1 Black-Scholes 편미분방정식의 공식

이 절에서는 Black-Scholes 편미분방정식의 해를 구한다.[1]

[1] 이 절에서 더 자세한 내용은 **금융증권을 위한 블랙숄즈의 편미분방정식(김완세 옮김)**의 책을 참고하길 바란다.

주어진 초기조건 $y(0) = 10$을 적용하면,

$$y^2 = -3x^2 + 100. \tag{6.13}$$

다음은 Fourier Transform에 대해서 알아보자.

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left[\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(v) e^{-iwv} dv \right] e^{iwx} dw. \tag{6.14}$$

여기서 괄호 안에 있는 w에 대한 함수를 $\hat{f}(w)$라 하고 함수 f의 Fourier Transform 이라고 한다.

$$\hat{f}(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(v) e^{-iwv} dv. \tag{6.15}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(w) e^{iwx} dw. \tag{6.16}$$

다음과 같은 예제 문제를 생각해보자.

예제 1 함수 $f(x)$ 가 다음과 같이 주어졌을때 f의 Fourier Transform 을 구하시오.

$$f(x) = 1 \text{ if } |x| < 1 \text{ and } f(x) = 0 \text{ otherwise.} \tag{6.17}$$

[해답]

식(6.15)을 이용하여 적분하면, 다음 결과를 얻을 수 있다.

$$\hat{f}(w) = \frac{1}{\sqrt{2\pi}} \int_{1}^{1} e^{-iwx} dx = \frac{1}{\sqrt{2\pi}} \cdot \frac{e^{-iwx}}{-iw} \Big|_{-1}^{1} = \frac{1}{-iw\sqrt{2\pi}} (e^{-iw} - e^{iw}).$$

1번째 변수변환

$$x = \log\frac{S}{E} + \left(r - \frac{\sigma^2}{2}\right)(T-t) \tag{6.19}$$

$$\tau = T - t \tag{6.20}$$

2개의 변수 x와 τ를 사용하여, $V(S,t)$를 다음과 같이 표현하자.

$$V(S,t) = e^{-r\tau}u(x,\tau),$$

여기서 함수 $V(S,t)$를 이와 같이 표현하면 블랙 숄즈의 편미분방정식을 매우 간단한 편미분방정식으로 고쳐 쓸 수 있게 된다. $V_S(S,t)$, $V_{SS}(S,t)$, $V_t(S,t)$를 각각 계산하여 보자. 연쇄법칙(chain rule)을 사용하여

$$\begin{aligned}
V_S(S,t) &= V_x(S,t)x_S + V_\tau(S,t)\tau_S = \frac{\partial}{\partial x}\left(e^{-r\tau}u(x,\tau)\right)x_S \\
&= e^{-r\tau}u_x(x,\tau)S^{-1}. \\
V_{SS}(S,t) &= \frac{\partial}{\partial S}\left(e^{-r\tau}u_x(x,\tau)S^{-1}\right) = e^{-r\tau}\left(u_{Sx}(x,\tau)S^{-1} - u_x(x,\tau)S^{-2}\right) \\
&= \frac{e^{-r\tau}}{S^2}\left(u_{Sx}(x,\tau)S - u_x(x,\tau)\right) \\
&= \frac{e^{-r\tau}}{S^2}\left(\left[u_{xx}(x,\tau)x_S + u_{\tau x}(x,\tau)\tau_S\right]S - u_x(x,\tau)\right) \\
&= \frac{e^{-r\tau}}{S^2}\left(u_{xx}(x,\tau) - u_x(x,\tau)\right).
\end{aligned}$$

$$\begin{aligned}
V_t(S,t) &= V_x(S,t)x_t + V_\tau(S,t)\tau_t \\
&= \frac{\partial}{\partial x}\left(e^{-r\tau}u(x,\tau)\right)x_t - \frac{\partial}{\partial \tau}\left(e^{-r\tau}u(x,\tau)\right) \\
&= e^{-r\tau}u_x(x,\tau)\frac{\partial}{\partial t}\left(\log\frac{S}{E} + (r - \frac{\sigma^2}{2})(T-t)\right) \\
&\quad -\frac{\partial}{\partial \tau}\left(e^{-r\tau}u(x,\tau)\right) \\
&= e^{-r\tau}u_x(x,\tau)(\frac{\sigma^2}{2} - r) - \left(-re^{-r\tau}u(x,\tau) + e^{-r\tau}u_\tau(x,\tau)\right) \\
&= e^{-r\tau}\left((\frac{\sigma^2}{2} - r)u_x(x,\tau) + ru(x,\tau) - u_\tau(x,\tau)\right).
\end{aligned}$$

c를 임의의 상수라 하고, 이제 위 식을 적분하면 다음의 식을 얻게 된다.

$$\begin{aligned}\int \frac{1}{\widehat{u}(\lambda,\tau)}\partial\widehat{u}(\lambda,\tau) + \int \frac{\sigma^2\lambda^2}{2}\partial\tau &= 0\\ \ln\widehat{u}(\lambda,\tau) + \frac{\sigma^2\lambda^2}{2}\tau &= c\\ \widehat{u}(\lambda,\tau) &= e^{-\frac{\sigma^2\lambda^2}{2}\tau+c}\end{aligned}$$

적분상수 c의 계산을 위해 초기조건을 적용해보자. 먼저, 초기조건 $u(x,0) = g(x)$을 $\widehat{u}$의 초기조건으로 바꿀 수 있다.

$$\widehat{u}(\lambda,0) = \widehat{g}(\lambda) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} g(x)e^{-i\lambda x}dx. \tag{6.27}$$

초기조건 (6.27)를 만족하는 상미분 방정식 (6.26)의 해가 다음과 같음을 쉽게 알 수 있다.

$$\widehat{u}(\lambda,\tau) = \widehat{g}(\lambda)e^{-\frac{\sigma^2\lambda^2\tau}{2}}. \tag{6.28}$$

식 (6.28)에 역 Fourier 변환을 적용해서 $u(x,\tau)$의 해를 구할 수 있다.

$$\begin{aligned}u(x,\tau) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\widehat{u}(\lambda,\tau)e^{i\lambda x}d\lambda = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\widehat{g}(\lambda)e^{i\lambda x-\frac{\sigma^2\lambda^2\tau}{2}}d\lambda\\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\left[\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}g(y)e^{-i\lambda y}dy\right]e^{i\lambda x-\frac{\sigma^2\lambda^2\tau}{2}}d\lambda\\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty}g(y)\left[\int_{-\infty}^{\infty}e^{-i\lambda(x-y)-\frac{\sigma^2\lambda^2\tau}{2}}d\lambda\right]dy.\end{aligned} \tag{6.29}$$

여기서 첫 번째 등호는 역 Fourier 변환에 의해서 두 번째 등호는 식 (6.28)에 의해서 성립한다. 식(6.29)의 우변의 적분을 계산하기 위해서 다음 변수를 정의하자.

$$\begin{aligned}I(\alpha) &= \int_{-\infty}^{\infty}e^{-\frac{\sigma^2\lambda^2\tau}{2}}e^{-i\lambda\alpha}d\lambda = \int_{-\infty}^{\infty}e^{-\frac{\sigma^2\lambda^2\tau}{2}}\left(\cos(\alpha\lambda) - i\sin(\alpha\lambda)\right)d\lambda\\ &= \int_{-\infty}^{\infty}e^{-\frac{\sigma^2\lambda^2\tau}{2}}\cos(\alpha\lambda)d\lambda = 2\int_{0}^{\infty}e^{-\frac{\sigma^2\lambda^2\tau}{2}}\cos(\lambda\alpha)d\lambda.\end{aligned} \tag{6.30}$$

기함수와 우함수의 성질을 이용하여 위의 식(6.30)으로 정리할 수 있다. 이

이 때 경계조건은 다음과 같이 쓸 수 있다.

$$g(y) = g(x + \sigma\sqrt{\tau}v) = \begin{cases} E(e^{x+\sigma\sqrt{\tau}v} - 1) & \text{if } \ v \geq \frac{-x}{\sigma\sqrt{\tau}} \\ 0 & \text{else} \end{cases}$$

2번째 변수변환을 하게 되면,

$$u(x,\tau) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(y) e^{-\frac{v^2}{2}} dv.$$

$g(y)$의 경계조건에 따라 $v < -\frac{x}{\sigma\sqrt{\tau}}$인 부분의 적분 값들은 모두 0의 값을 갖게 되므로, $v \geq \frac{-x}{\sigma\sqrt{\tau}}$의 부분의 적분 값만 생각하면 된다. $g(y)$의 경계조건에 주목하면,

$$\begin{aligned} u(x,\tau) &= \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} g(y) e^{-\frac{v^2}{2}} dv = \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} (Ee^{x+\sigma\sqrt{\tau}v} - E) e^{-\frac{v^2}{2}} dv \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} Ee^{x+\sigma\sqrt{\tau}v} e^{-\frac{v^2}{2}} dv - \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} Ee^{-\frac{v^2}{2}} dv \qquad (6.38) \end{aligned}$$

첫 번째 변수변환(6.20)에 의해 식(6.38)의 첫 번째 항은 다음을 만족한다.

$$\begin{aligned} u(x,\tau) &= \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} Se^{r\tau - \frac{\sigma^2}{2}\tau} e^{\sigma\sqrt{\tau}v} e^{-\frac{v^2}{2}} dv - \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} Ee^{-\frac{v^2}{2}} dv \\ &= \frac{Se^{r\tau}}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} e^{-\frac{1}{2}(v-\sigma\sqrt{\tau})^2} dv - \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} Ee^{-\frac{v^2}{2}} dv. \qquad (6.39) \end{aligned}$$

여기서 $z = v - \sigma\sqrt{\tau}$로 변수변환을 하게 되면,

$$\begin{aligned} u(x,\tau) &= Se^{r\tau} \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}} - \sigma\sqrt{\tau}}^{+\infty} e^{-\frac{z^2}{2}} dz - \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sigma\sqrt{\tau}}}^{+\infty} Ee^{-\frac{v^2}{2}} dv \\ &= Se^{r\tau} N\left(\frac{x}{\sigma\sqrt{\tau}} + \sigma\sqrt{\tau}\right) - EN\left(\frac{x}{\sigma\sqrt{\tau}}\right), \end{aligned}$$

여기서 $N(x)$는 누적표준정규분포함수(the cumulative standard normal distribution function)를 뜻하며 다음 식을 만족한다.

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} dz$$

```
public:
enum enumEC{E=1, T, SIGMA, R, S};
eurocall() { init(0, 0, 0.0, 0.0, 0.0); }
eurocall(double nS, int nE, double nT, double nSigma, double nR) {
init(nS, nE, nT, nSigma, nR);}
eurocall(const eurocall &ec) { init(ec.m_nS, ec.m_nE, ec.m_nT,
ec.m_nSigma, ec.m_nR); }
virtual ~eurocall(){}
void init(double nS, int nE, double nT, double nSigma, double nR);
template <typename V>
void SetData(V nData, int nType);
int GetEData(void) { return m_nE; }
double GetTData(void) { return m_nT; }
double GetSigmaData(void) { return m_nSigma; }
double GetRData(void) { return m_nR; }
double GetSData(void) { return m_nS; }
void ViewData(eurocall &obj) const;
double GetOptionPrice(double nS, int nE, double nT, double nSigma,
double nR) const;
double normcdf(double X) const;
private:
int  m_nE;
double m_nT, m_nSigma, m_nR, m_nS;
};
void eurocall::init(double nS, int nE, double nT, double nSigma, double
nR)
{
SetData(nS, S);
SetData(nE, E);
SetData(nT, T);
SetData(nSigma, SIGMA);
SetData(nR, R);
}
template <typename V>
void eurocall::SetData(V nData, int nType)
{
```

```
{
cout <<"S : " << obj.GetSData() <<endl;
cout <<"E : " << obj.GetEData() <<endl;
cout <<"T : " << obj.GetTData() <<endl;
cout <<"Sigma : " << obj.GetSigmaData() <<endl;
cout <<"R : " << obj.GetRData() <<endl;
}
int main()
{
eurocall euro(240,250,(double)2/12,0.38,0.06);
double callPrice = euro.GetOptionPrice(euro.GetSData(), euro.GetEData(),
euro.GetTData(), euro.GetSigmaData(), euro.GetRData());
euro.ViewData(euro);
cout << "Call Price : " << callPrice << endl;
cout << "******************" << endl;
euro.SetData(300, eurocall::S);
euro.SetData(0.1, eurocall::R);
callPrice = euro.GetOptionPrice(euro.GetSData(), euro.GetEData(),
euro.GetTData(), euro.GetSigmaData(), euro.GetRData());
euro.ViewData(euro);
cout << "Call Price : " << callPrice << endl;
return 0;
}
```

위의 C++ 코드eurocall.cpp을 실행하면 다음의 결과를 얻는다.

1.2 옵션가격의 성질

위에서 언급하였지만 옵션가격은 다섯 가지의 변수에 대한 함수이다. 이 절에서는 이 변수들이 각각 옵션가격에 어떠한 영향을 미치는지 확인해보도록 하자.

1.2.1 기초자산가격(S)

위의 예제를 이용하여 기초자산가격(S) 이외의 다른 네 개의 변수는 동일한 값을 가지고 있다고 가정할 때, 옵션가격은 기초자산가격(S)에 대하여

```
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}
vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
class option_s
{
public:
option_s() { init(0, 0, 0.0, 0.0, 0.0); };
option_s(double nS, double nE, double nT, double nSigma, double nR)
{ init(nS, nE, nT, nSigma, nR);}
option_s(const option_s &data)
{ init(data.m_S, data.m_E, data.m_T, data.m_Sigma, data.m_R); }
~option_s(){}
void GetOptionPrice(void);
void init(double nS, double nE, double nT, double nSigma, double nR);
void WriteData(vector<double> X,vector<double> CP);
private:
double m_Sigma,m_T,m_R,m_E,m_S;
};
void option_s::WriteData(vector<double> X,vector<double> CP)
{
ofstream output;
output.open("option_s_S_Data.dat");
output<<fixed<<setprecision(6);
```

```
int main()
{
option_s s1(400,250,(double)2/12,0.38,0.06);
s1.GetOptionPrice();

}
```

위의 결과로 컴파일한 프로젝트가 있는 폴더 안에 option_s_S_Data.dat 과 option_s_CP_Data.dat 두 개의 파일이 생성 된다. 지금의 파일들은 data파일의 형태이므로 눈으로 확인 하는 것이 쉽지 않다. 지금의 data파일을 MATLAB을 통하여 그래프로 표현할수 있는 m-파일을 제시 한다. 이하 책에 쓰여진 C++코드가 data파일을 생성하는 경우 그에 따르는 파일은 data파일을 plot하는 m-파일이다.

```
clc; clf; clear;
S = load('option_s_S_Data.dat');
CP = load('option_s_CP_Data.dat');

plot(S, CP,'k*-')
xlabel('S','fontsize',20);
ylabel('V','fontsize',20,'rotation',0);
set(gca,'fontsize',20)
```

위의 코드에 의한 결과로 얻어진 그림 6.3의 우측의 그림은 좌측의 그림을 확대한 것이다.

1.2.2 행사가격(E)

행사가격(E)에 따라 옵션가치의 변화를 다음의 그림 6.4를 통해 확인해 보자. 먼저, option_ex.cpp는 행사가격에 따른 옵션가격을 구하는 C++ 코드이다.

```
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
enum enumFILE {IN=0x01, OUT, ATE=0x04, APP=0x08, FILE_MAX};
class option_ex
{
public:
option_ex() { init(0.0, 0.0, 0.0, 0.0, 0.0); };
option_ex(double nS, double nE, double nT, double nSigma, double nR)
{ init(nS, nE, nT, nSigma, nR);}
option_ex(const option_ex &data)
{ init(data.m_S, data.m_E, data.m_T, data.m_Sigma, data.m_R); }
virtual ~option_ex(){}
void GetOptionPrice(int nType);
void init(double nS, double nE, double nT, double nSigma, double nR);
void WriteData(vector<double> X,vector<double> CP,int nType);
private:
double m_Sigma,m_T,m_R,m_E,m_S;
};
void option_ex::WriteData(vector<double> X,vector<double> CP,int nType)
{
ofstream output;
output.open("option_ex_S_Data.dat");
output<<fixed<<setprecision(6);
for(unsigned int i=0;i<X.size();i++)
{
output<<setw(12)<<showpoint<<X[i];
}
output.clear();
output.close();
output.open("option_ex_CP_Data.dat",nType);
for(unsigned int i=0;i<CP.size();i++)
```

```
}
```

다음은 생성된 data파일을 읽는 m-파일이다.

```
clc; clf; clear;
S = load('option_ex_S_Data.dat');
CP = load('option_ex_CP_Data.dat')';

plot(S,CP(:,1),'k*-',S, CP(:,2),'ko-',S,CP(:,3),'k^-')
legend('E = 200','E = 250','E = 300',2)
xlabel('S','fontsize',20)
ylabel('V','fontsize',20,'rotation',0)
set(gca,'fontsize',20)
```

위의 코드를 실행시켜보면 다음의 결과를 얻을 수 있다. 행사가격을 제외한 모든 변수들을 고정하였을 경우 행사가격(E)이 증가할수록 옵션가격은 감소하게 된다.

1.2.3 이자율(r)

그림 6.5에서 볼 수 있듯이 옵션가격은 이자율(r)에 대한 증가함수로서 이자율(r)이 증가할 수록 옵션가격도 증가하게 된다. option_r.cpp는 이자율에 따른 옵션가격을 구하는 C++ 코드로서 그림 6.5을 얻을 수 있다.

```
//////////////////////+option_r.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)
```

```
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
enum enumFILE {IN=0x01, OUT, ATE=0x04, APP=0x08, FILE_MAX};
class option_r
{
public:
option_r() { init(0, 0, 0.0, 0.0, 0.0); };
option_r(double nS, double nE, double nT, double nSigma, double nR)
{ init(nS, nE, nT, nSigma, nR);}
option_r(const option_r &data)
{ init(data.m_S, data.m_E, data.m_T, data.m_Sigma, data.m_R); }
virtual ~option_r(){}
void GetOptionPrice(int nType);
void init(double nS, double nE, double nT, double nSigma, double nR);
void WriteData(vector<double> X,vector<double> CP,int nType);
private:
double m_Sigma,m_T,m_R,m_E,m_S;
};
void option_r::WriteData(vector<double> X,vector<double> CP,int nType)
{

ofstream output;
output.open("option_r_S_Data.dat");
output<<fixed<<setprecision(6);
for(unsigned int i=0;i<X.size();i++)
{
output<<setw(12)<<showpoint<<X[i];
}
output<<endl;
output.clear();
output.close();
output.open("option_r_CP_Data.dat",nType);
for(unsigned int i=0;i<CP.size();i++)
```

```
option_r r3(400,250,(double)2/12,0.38,0.4);
r1->GetOptionPrice(OUT);
r2.GetOptionPrice(APP);
r3.GetOptionPrice(APP);
delete[] r1;

}
```

다음은 생성된 data파일을 읽는 m-파일이다.

```
clc; clf; clear;
S = load('option_r_S_Data.dat');
CP = load('option_r_CP_Data.dat')';

plot(S,CP(:,1),'k*-',S, CP(:,2),'ko-',S,CP(:,3),'k^-')
legend('r = 0.00','r = 0.20','r = 0.40',2)
xlabel('S','fontsize',20)
ylabel('V','fontsize',20,'rotation',0)
set(gca,'fontsize',20)
```

1.2.4 잔존기간(T)

option_T.m는 잔존기간에 따라 옵션가격을 구하는 C++ 코드로서 실행하면 그림 6.6를 얻게 된다.

```
//////////////////////+option_t.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
```

```
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
enum enumFILE {IN=0x01, OUT, ATE=0x04, APP=0x08, FILE_MAX};
class option_t
{
public:
option_t() { init(0, 0, 0.0, 0.0, 0.0); };
option_t(double nS, double nE, double nT, double nSigma, double nR)
{ init(nS, nE, nT, nSigma, nR);}
option_t(const option_t &data)
{ init(data.m_S, data.m_E, data.m_T, data.m_Sigma, data.m_R); }
virtual ~option_t(){}
void GetOptionPrice(int nType);
void init(double nS, double nE, double nT, double nSigma, double nR);
void WriteData(vector<double> X,vector<double> CP,int nType);
private:
double m_Sigma,m_T,m_R,m_E,m_S;
};
void option_t::WriteData(vector<double> X,vector<double> CP,int nType)
{
ofstream output;
output.open("option_t_S_Data.dat");
output<<fixed<<setprecision(6);
for(unsigned int i=0;i<X.size();i++)
{
output<<setw(12)<<showpoint<<X[i];
}
output.clear();
output.close();
output.open("option_t_CP_Data.dat",ios::app);
for(unsigned int i=0;i<CP.size();i++)
{
```

```
t1->GetOptionPrice(OUT);
t2.GetOptionPrice(APP);
t3.GetOptionPrice(APP);
delete[] t1;
}
```

다음은 생성된 data파일을 읽는 m-파일이다.

```
clc; clf; clear;
S = load('option_t_S_Data.dat');
CP = load('option_t_CP_Data.dat')';

plot(S,CP(:,1),'k*-',S, CP(:,2),'ko-',S,CP(:,3),'k^-')
legend('T = 1/12','T = 5/12','T = 10/12',2)
xlabel('S','fontsize',20)
ylabel('V','fontsize',20,'rotation',0)
set(gca,'fontsize',20)
```

그림 6.6에서 확인 할 수 있듯이 잔존기간이 증가할수록 옵션가격도 증가한다.

1.2.5 변동성(σ)

옵션 가격은 변동성(σ)에 대한 증가함수이다. 이를 확인하기 위해 C++ 코드option_sig.cpp을 실행해보자.

```
//////////////////////+option_sig.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
```

```
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
enum enumFILE {IN=0x01, OUT, ATE=0x04, APP=0x08, FILE_MAX};
class option_sig
{
public:
option_sig() { init(0, 0, 0.0, 0.0, 0.0); };
option_sig(double nS, double nE, double nT, double nSigma, double nR)
{ init(nS, nE, nT, nSigma, nR);}
option_sig(const option_sig &data)
{ init(data.m_S, data.m_E, data.m_T, data.m_Sigma, data.m_R); }
virtual ~option_sig(){}
void GetOptionPrice(int nType);
void init(double nS, double nE, double nT, double nSigma, double nR);
void WriteData(vector<double> X,vector<double> CP,int ntype);
private:
double m_Sigma,m_T,m_R,m_E,m_S;
};
void option_sig::WriteData(vector<double> X,vector<double> CP,int nType)
{
ofstream output;
output.open("option_sig_S_Data.dat");
output<<fixed<<setprecision(6);
for(unsigned int i=0;i<X.size();i++)
{
output<<setw(12)<<showpoint<<X[i];
}
output.clear();
output.close();
output.open("option_sig_CP_Data.dat",nType);
for(unsigned int i=0;i<CP.size();i++)
{
```

다음은 생성된 data파일을 읽는 m-파일이다.

```
clc; clf; clear;
S = load('option_sig_S_Data.dat');
CP = load('option_sig_CP_Data.dat')';

plot(S,CP(:,1),'k*-',S, CP(:,2),'ko-',S,CP(:,3),'k^-')
legend('\sigma = 0.10','\sigma = 0.38','\sigma = 0.70',2)
xlabel('S','fontsize',20)
ylabel('V','fontsize',20,'rotation',0)
set(gca,'fontsize',20)
```

위의 C++ 코드에 따른 결과는 그림 6.7에서 확인할 수 있다.

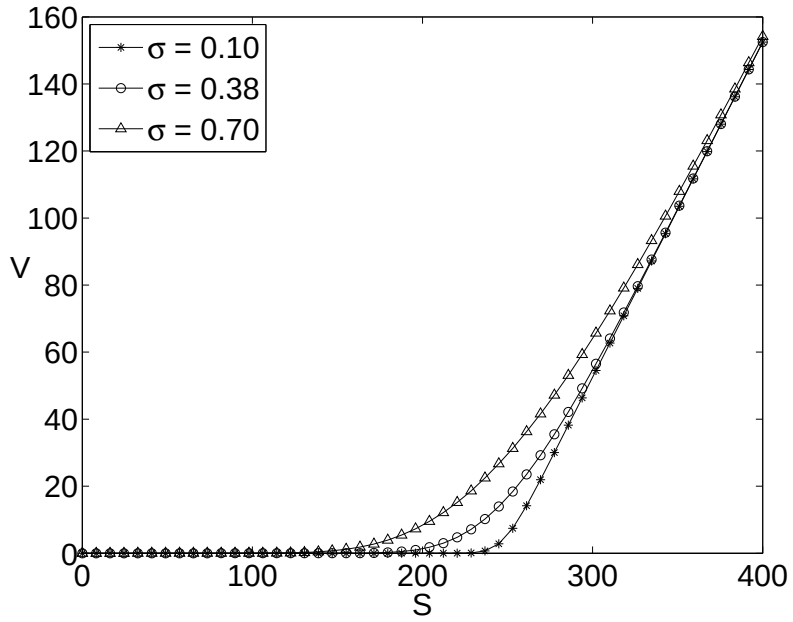


그림 6.7: 변동성(σ)에 따른 옵션가격의 변화

제 7 장

변동성 추정

변동성(volatility)은 옵션의 만기까지의 기초자산가격 변화율의 분포의 표준편차라고 정의할 수 있다. 기조자산이 일정기간동안 어느정도 움직이는지를 나타내는 수치로 과거정보를 바탕으로 일정기간 동안의 기초자산가격 변화율의 표준편차는 역사적변동성이라하고, 옵션의 시장가격에 내재된 변동성을 내재변동성이라고 정의한다.

제 1 절 내재 변동성

Black-Scholes 옵션가격결정모형에서 산출된 이론가격이 시장가격과 같아지도록 하는 변동성을 내재변동성이라 한다. 이러한 내재변동성은 실제 시장에서 거래하는 사람들이 느끼는 체감 변동성이라고 할 수 있다. 옵션의 이론가는 기초자산가격, 행사가격, 잔존기간, 변동성, 이자율로 결정되어지는데 이중 변동성을 변수로 놓고 역산하여 계산한 값이 내재변동성이다. 이러한 내재변동성은 일반적으로 수치해석적 방법을 적용하여 계산한다. 이 장에서는 콜옵션의 내재변동성을 뉴튼 랩슨법을 이용하여 구해보기로 하자.

여기서 σ_i는 추정변동성, V_m은 옵션의 시장가격, $\frac{\partial V}{\partial \sigma_i} = S\sqrt{T}N'(d_1)$[1]는 베가(Vega, 변동성에 따른 옵션가치의 민감도, 자세한 내용은 제 7장의 유한차분법의 Greek참고)를 나타낸다. 다음은 내재변동성을 구하는 간단한 예제이다. 기초자산가격 $S = 100$, 행사가격 $E = 100$, 이자율 $r = 0.05$, 만기시간은 $T = 1.0$, 그리고 옵션의 시장가격 $V_m = 20$으로 가정하고 초기 값은 0.2로 설정하였고, 수치 해석적 방법으로 해를 구할 때 필요한 오차 한계치는 $1.0e-6$로 하였다. `Newton_implied_vol.cpp`는 위의 식을 이용하여 내재변동성을 구한 C++ 코드이다.

```
//////////////////////+Newton_implied_vol.cpp+/////////////////////
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

double normcdf(double X)
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937,
a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}
double normpdf(double X, double mu, double sigma)
{
```

[1]위의 식에서 사용되는 d_1은 다음과 같다.

$$d_1 = \frac{\ln(S/E) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

```
return 0;
}
```

출력결과는 다음과 같다.

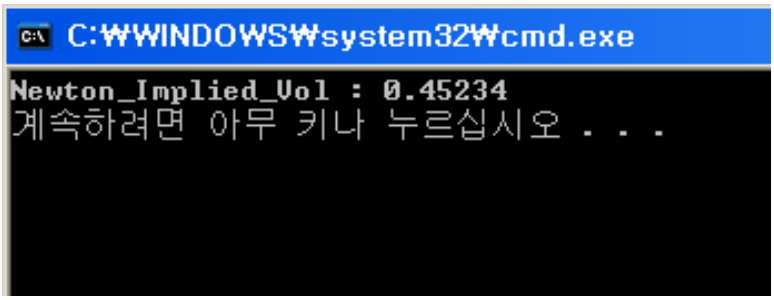


그림 7.2: 뉴튼랩슨법 해 찾기

연습문제 1.1

1. $f(x) = e^{2x} - e^x - 2 = 0$은 구간 $[0, 1]$에서 근을 갖는다. Newton-Rapson 방법을 사용하여 근을 구하여라.

제 8 장

유한 차분법
(Finite Difference Method)

제 1 절 개요

유한차분법은 미분방정식 (differential equation)을 차분방정식 (difference equation)으로 이산화 시켜서 수치적인 해를 구하는 방법이다. 이 장에서는 유한차분법을 사용하여 열방정식 (heat equation)과 블랙 숄즈 편미분 방정식의 근사해를 구할 것이다. 먼저 Taylor의 정리[1]를 바탕으로 하고 있는 유한차분법의 기본 원리를 살펴보자. Taylor의 정리를 이용하면 함수 $u(x+h,t)$는 다음과 같이 (x,t) 에서의 u 함수값과 미분값들의 무한급수로 나타낼 수 있다.

$$u(x+h,t) = u(x,t) + u_x(x,t)h + \frac{u_{xx}(x,t)}{2}h^2 + \frac{u_{xxx}(x,t)}{3!}h^3 + \cdots \quad (8.1)$$

$u_x(x,t)$에 대해서 정리하면, 1차미분에 대한 차분식을 얻는다.

$$u_x(x,t) = \frac{u(x+h,t) - u(x,t)}{h} + O(h). \quad (8.2)$$

[1]Taylor 정리: 함수 $f(x)$가 $x = x_0$에서 n번 미분가능하다고 하자.

$$p_n(x) = u(x_0) + u'(x_0)(x-x_0) + \frac{u''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{u^{(n)}(x_0)}{n!}(x-x_0)^n$$

을 $x = x_0$에서 $u(x)$의 n번째 Taylor 다항식이라 한다.

제 2 절 열 방정식에 대한 유한 차분법

유한차분법을 이용하여 열방정식을 풀어보기로 하자. 열방정식은 식(8.8)과 같은 편미분 방정식이다.

$$u_t(x,t) = u_{xx}(x,t), \qquad 0 < x < 1, \qquad t > 0. \tag{8.8}$$

이 때 경계조건은 $u(0,t) = u(1,t) = 0 \ \ (t > 0)$이고 초기조건은 $u(x,0) = \sin(\pi x) \ \ (0 \le x \le 1)$을 만족한다. 해석해는 $u(x,t) = \sin(\pi x)\mathrm{e}^{-\pi^2 t}$이며 그림8.2와 그림8.3처럼 그래프로 나타낼 수 있다. 이 방정식에 대한 근사해를 명시적, 함축적, 그리고 크랭크-니콜슨 유한차분법을 이용하여 구해보자.

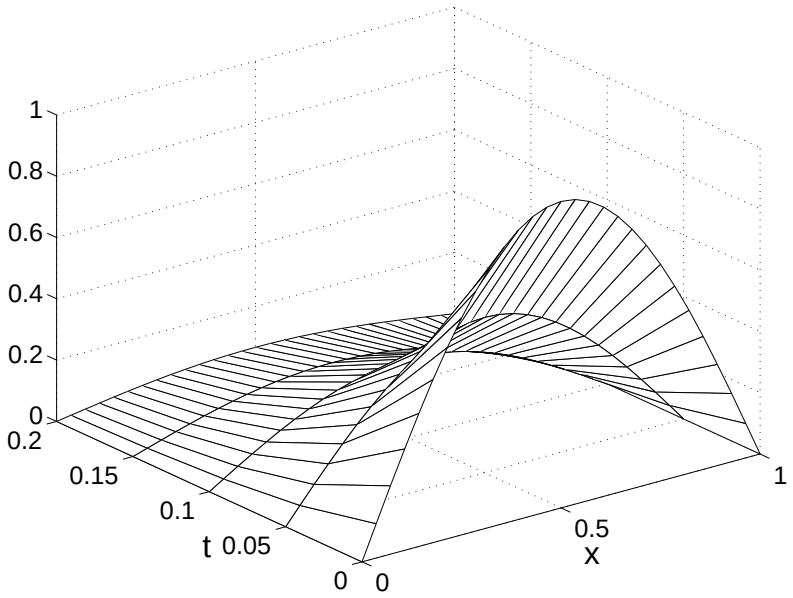


그림 8.2: 열방정식의 해석해

2.1 명시적 (Explicit) 유한 차분법

먼저 정수 $N_x > 0$을 선택하고 $h = 1/(N_x - 1)$이라 정의하면, 식(8.3)와 (8.7)을 이용하여, 열방정식(8.8)에 대해 다음과 같이 유한차분법을 적용할

을 명시적이라 부르는 이유이다. `heatex.cpp`은 $\alpha = 0.45$, $N_x = 30$일 때, 시간에 따른 열방정식의 해를 나타낸 C++ 코드이다.

```
//////////////////////+heatex.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
class heatex
{
private:
double m_alpha, m_T;
int m_Nt,m_Nx;
public:
heatex(){init(.0,0,.0);};
heatex(const heatex &data)
{init(data.m_alpha,data.m_Nx,data.m_T);};
heatex(double nalpha, int nNx, double nT) :
  m_alpha(nalpha), m_Nx(nNx),m_T(nT){}
  ~heatex(){};
  void init(double nalpha, int nNx, double nT);
  void GetNumericalSolution();
  void WriteData(vector<double> S, vector<vector<double>>
CP,vector<vector<double>> CPP);
};
```

```
output.close();
}

void heatex::GetNumericalSolution()
{
vector<double> x=linspace(0,1,m_Nx);
double h=x[1]-x[0],k= m_alpha*pow(h,2);
m_Nt=(int)floor(m_T/k+0.5);
vector<vector<double>> u(m_Nx, vector<double>(m_Nt+1)),exu(m_Nx,
vector<double>(m_Nt+1));
for (int i=0;i<m_Nx;i++)
u[i][0]=sin(PI*x[i]);
exu=u;
for (int n=0; n<m_Nt;n++)
{
for (int i=1; i<m_Nx-1;i++)
{
exu[i][n+1]=sin(PI*x[i])*exp(-pow(PI,2)*(k*(n+1)));
u[i][n+1]=u[i][n]+m_alpha*(u[i-1][n]-2.0*u[i][n]+u[i+1][n]);
}
}
WriteData(x,u,exu);
}

int main()
{
heatex ex(0.45,30,0.125);
ex.GetNumericalSolution();
return 0;
}
```

다음은 data파일을 읽는 m-파일이다.

```
clc; clf; clear;
```

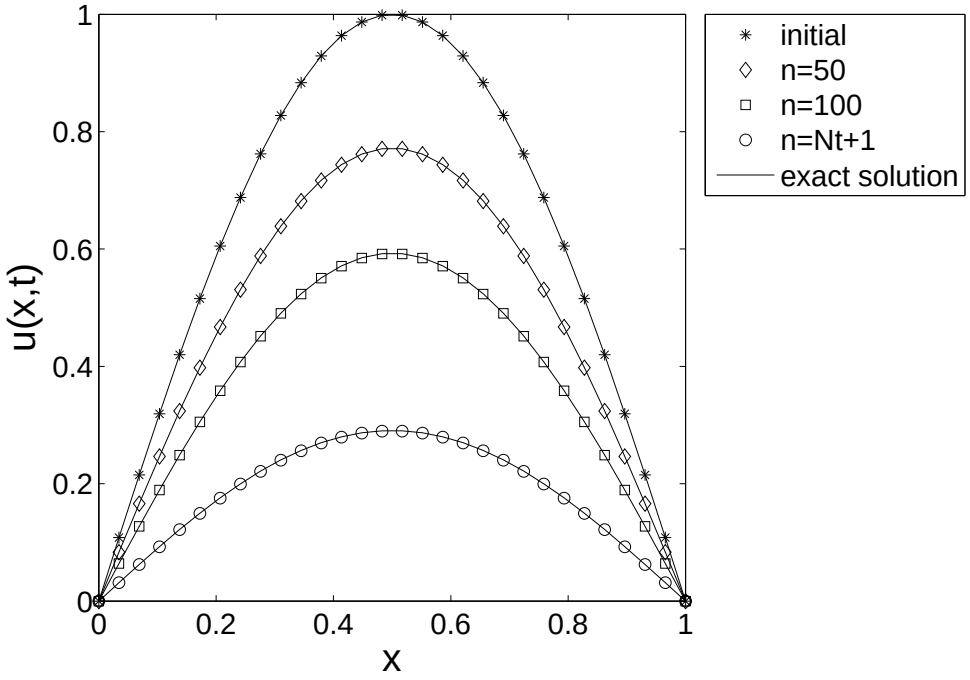


그림 8.4: $\alpha = 0.45$인 안정한 상태

$$0 \leq \alpha \sin^2 \frac{\beta h}{2} \leq \frac{1}{2}. \tag{8.12}$$

따라서, 양유한차분해가 안정적이기 위한 필요충분조건은 다음과 같다.

$$0 < \alpha \leq \frac{1}{2}. \tag{8.13}$$

그림 8.4에서 $\alpha = 0.45$일때 계산된 값은 정확한 해에 가까워지지만, 그림 8.5에서 보듯이 $\alpha = 0.55$일때 계산된 값은 정확한 해로 수렴하지 않는다. 즉 α의 값이 커짐에 따라 구한 해가 불안정할 수 있다는 점이 명시적 유한차분법의 단점이다.

을 얻게 된다. (8.15)는 다음과 같은 선형 시스템 (linear system)으로 나타낼 수 있다.

$$\begin{pmatrix} 1+2\alpha & -\alpha & 0 & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & & 0 \\ 0 & -\alpha & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -\alpha \\ 0 & 0 & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ \vdots \\ u_{N_x-1}^{n+1} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha u_1^{n+1} + u_2^n \\ u_3^n \\ \vdots \\ \vdots \\ u_{N_x-1}^n + \alpha u_{N_x}^{n+1} \end{pmatrix} = \begin{pmatrix} b_2^n \\ b_3^n \\ \vdots \\ \vdots \\ b_{N_x-1}^n \end{pmatrix}. \tag{8.15}$$

(8.15)은 좀 더 압축된 형태인

$$\mathbf{A}\boldsymbol{u}^{n+1} = \boldsymbol{b}^n \tag{8.16}$$

으로 나타낼 수 있다. 여기서 $\boldsymbol{u}^{n+1}$와 $\boldsymbol{b}^n$는 $(N_x-2)-$차원의 벡터들인

$$\boldsymbol{u}^{n+1} = (u_2^{n+1}, \cdots, u_{N_x-1}^{n+1})^T, \quad \boldsymbol{b}^n = \boldsymbol{u}^n + \alpha(u_1^n, 0, \cdots, 0, u_{N_x}^{n+1})^T$$

을 뜻하며, $\mathbf{A}$은 (8.15)에서 주어진 $(N_x-2)-$정방대칭행렬을 뜻한다. $\alpha \geq 0$에 대하여 $\mathbf{A}$은 가역 (invertible)이므로

$$\boldsymbol{u}^{n+1} = \mathbf{A}^{-1}\boldsymbol{b}^n \tag{8.17}$$

이다. 그러므로 $\boldsymbol{u}^n$과 경계조건에 의해 구해지는 $\boldsymbol{b}^n$에 의해 $\boldsymbol{u}^{n+1}$를 찾을 수 있다. 초기조건은 $\boldsymbol{u}^1$에 의해 결정되므로, 각각의 $\boldsymbol{u}^{n+1}$을 순차적으로 구할 수 있다.

치하게 되고 d_2와 b_2의 값도 다음과 같이 변하게 된다.

$$d_2 = d_2 - \frac{a_1}{d_1}c_1, \qquad b_2 = b_2 - \frac{a_1}{d_1}b_1$$

하지만 c_2는 변하지 않는다.

$$\begin{pmatrix} d_1 & c_1 & & & & & & \\ 0 & d_2 - \frac{a_1c_1}{d_1} & c_2 & & & & & \\ & a_2 & d_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & a_{i-1} & d_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & a_{N_x-2} & d_{N_x-1} & c_{N_x-1} \\ & & & & & & a_{N_x-1} & d_{N_x} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{N_x-1} \\ x_{N_x} \end{pmatrix}$$

$$= \begin{pmatrix} b_1 \\ b_2 - \frac{a_1b_1}{d_1} \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x-1} \\ b_{N_x} \end{pmatrix}.$$

이 과정을 반복하여 적용하면 각각의 단계에서 d_i와 b_i들은 다음과 같이 바뀌게 된다:

$$d_i = d_i - \frac{a_{i-1}}{d_{i-1}}c_{i-1}, \qquad b_i = b_i - \frac{a_{i-1}}{d_{i-1}}b_{i-1} \qquad (2 \le i \le N_x).$$

```
vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
class heatim
{
private:
int m_Nx, m_Nt, m_start, m_end;
double m_h, m_T,m_alpha, m_k ;
public:
heatim(int m_Nx,double m_T, double m_alpha);
~heatim(){};
void GetNumericalSolution();
void WriteData(vector<double> S, vector<vector<double>> CP);
};

heatim::heatim(int nNx, double nT, double nalpha)
{
m_Nx=nNx;m_T=nT;m_alpha=nalpha;m_start=0;m_end=1;
}
void heatim::GetNumericalSolution()
{
vector<double> x=linspace(m_start, m_end, m_Nx);
m_h=x[2]-x[1];
m_k=m_alpha*pow(m_h,2);
m_Nt=floor(m_T/m_k+0.5);
vector<vector<double>> u(m_Nx, vector<double>(m_Nt+1));
for (int i=0;i<m_Nx;i++)
u[i][0]=sin(PI*x[i]);
vector<double> dd(m_Nx-2),c(m_Nx-2),a(m_Nx-2);
for (int i=0; i<m_Nx-2;i++)
{
```

```
for (unsigned int j=0; j< CP[i].size(); j++) {
output<<setw(12)<<showpoint<<CP[i][j];
}
output<<endl;
}
output.clear();
output.close();
}
int main()
{
heatim im(12,0.1,2.0);
im.GetNumericalSolution();
return 0;
}
```

다음은 data 파일을 읽는 m-파일이다.

```
clc; clf; clear;

x = load('heatim_S_Data.dat');
u = load('heatim_CP_Data.dat');

plot(x,u,'ko-')
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20);
```

위의 코드 heatim.cpp를 실행하면 다음의 결과를 얻을 수 있다.

2.2.2 함축적 방법의 안정성 문제 - 폰 노이만 방법

명시적 방법의 안정성 문제와 같이 시간이 지남에 따라서

$$u_k^n = e^{i\beta kh}\xi^n \tag{8.19}$$

이 어떻게 성장하는가 보자. 식 (8.19)을 방정식 (8.15)에 대입을 하면 다음

는 미분 근사값 $u_i^{n+1/2}$을 이용한다. 시점 $n+1/2$에서 , 시간에 대한 1계 편미분을 구하면 다음과 같다.

$$u_t(x_i, t^{n+1/2}) = \frac{u_i^{n+1} - u_i^n}{k} + O(k^2) \tag{8.21}$$

또한 시점 $n+1/2$에서, 공간변수 x에 대한 2계 편미분은 시점 n와 $n+1$에서 2계 편미분 근사값을 평균해서 구한다.

$$\begin{aligned} u_{xx}(x_i, t^{n+1/2}) &= \frac{1}{2}\left(u_{xx}(x_i, t^n) + u_{xx}(x_i, t^{n+1})\right) + O(h^2) \qquad (8.22) \\ &= \frac{1}{2}\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2}\right) \\ &\quad + O(h^2). \end{aligned}$$

두 근사식 (8.21)와 (8.22)의 절단오차는 각각 $\mathcal{O}(k^2)$과 $\mathcal{O}(h^2)$으로 근사식의 정확도가 높기 때문에 많은 계산을 하지 않아도 수치분석에서 만족스러운 해를 얻을 수 있다. 식(8.21)와 (8.22)의 우변을 같게 하면 다음과 같은 크랭크 니콜슨 식을 얻을 수 있다.

$$-\alpha u_{i-1}^{n+1} + 2(1+\alpha)u_i^{n+1} - \alpha u_{i+1}^{n+1} = \alpha u_{i-1}^n + 2(1-\alpha)u_i^n + \alpha u_{i+1}^n, \tag{8.23}$$

```
#define PI 4.0*atan(1.0)

vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
class heatCN
{
private:
int m_Nx, m_Nt, m_start, m_end;
double m_h, m_T,m_alpha, m_k ;
public:
heatCN(int m_Nx,double m_T, double m_alpha);
~heatCN(){};
void GetNumericalSolution();
void WriteData(vector<double> S, vector<vector<double>> CP);
};

heatCN::heatCN(int nNx, double nT, double nalpha)
{
m_Nx=nNx;m_T=nT;m_alpha=nalpha;m_start=0;m_end=1;
}
void heatCN::GetNumericalSolution()
{
vector<double> x=linspace(m_start, m_end, m_Nx);
m_h=x[2]-x[1];
m_k=m_alpha*pow(m_h,2);
m_Nt=floor(m_T/m_k+0.5);
vector<vector<double>> u(m_Nx, vector<double>(m_Nt+1));
for (int i=0;i<m_Nx;i++)
{
u[i][0]=sin(PI*x[i]);
```

```
output<<setw(12)<<showpoint<<S[i];
output.clear();
output.close();
output.open("heatCN_u_Data.dat");
for(unsigned int n=0;n<CP[0].size();n++)
{
for (unsigned int i=0;i<S.size();i++)
{
output<<setw(12)<<showpoint<<CP[i][n];
}
output<<endl;
}
output.clear();
output.close();
}
int main()
{
heatCN CN(12,0.1,2.0);
CN.GetNumericalSolution();
return 0;
}
```

다음은 data파일을 읽는 m-파일이다.

```
clc; clf; clear;

x = load('heatCN_x_Data.dat');
u = load('heatCN_u_Data.dat');

plot(x,u,'ko-');
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
```

위의 코드 heatCN.cpp를 실행하면 다음의 결과를 얻을 수 있다.

ξ는 모든 양수 α와 모든 β에 대해서 $\frac{1}{4\alpha+1} \leq \xi \leq 1$을 만족한다. 따라서 식(8.15)은 무조건 안정적이다. 이는 그림 8.6로 확인할 수 있다.

2.4 수렴성 (convergence) 테스트

국소절단오차 (local truncation error)는 연속해가 노드점에서 수치적 방법을 만족하지 못하는 차이를 측정한 것이다. 수치 기법의 국소절단오차는 연속적인 문제의 정확한 해를 이산적인 수치기법에 대입함으로써 발생한다. $u(x_i, t^n)$는 열방정식의 정확한 해를 나타낸다. 다음은 정확한 해를 수치기법에 대입함으로써 명시적 유한차분법의 국소절단오차를 찾는 과정이다. 노드 (x_i, t^n)에서 국소절단오차는 다음과 같이 구한다.

$$T(x_i, t^n) = \frac{u(x_i, t^{n+1}) - u(x_i, t^n)}{k} - \frac{u(x_{i+1}, t^n) - 2u(x_i, t^n) + u(x_{i-1}, t^n)}{h^2}.$$

이제 노드(x_i, t^n)에서 테일러전개를 하면 각각의 항을 다음과 같이 나타낼 수 있다.

$$\begin{aligned} T(x_i, t^n) &= u_t(x_i, t^n) + \frac{k}{2} u_{tt}(x_i, t^n) + \mathcal{O}(k^2) \\ &\quad -u_{xx}(x_i, t^n) + \frac{h^2}{12} u_{xxxx}(x_i, t^n) + \mathcal{O}(h^4). \end{aligned}$$

여기서 $u(x_i, t^n)$은 열방정식을 만족하므로 다음이 성립한다.

$$\begin{aligned} T(x_i, t^n) &= \frac{k}{2} u_{tt}(x_i, t^n) + \frac{h^2}{12} u_{xxxx}(x_i, t^n) + \mathcal{O}(k^2) + \mathcal{O}(h^4) \\ &= \mathcal{O}(k) + \mathcal{O}(h^2). \end{aligned} \tag{8.28}$$

수치적 해가 수렴하기 위해 필요한 조건은 수치기법의 국소절단오차가 공간간격과 시간간격을 줄일수록 0에 근사해야 한다는 것이다. 이럴 경우에, 수치기법이 일관적 (consistent)이라고 한다. 정확도의 차수 (order of accuracy)는 절단오차항에서 h와 k의 승수의 차수로 정의된다. 절단오차항을 $\mathcal{O}(k^l + h^m)$로 가정하면 수치기법이 l차 시간 정확 (lth order time accurate)하고 m차 공간정확 (mth order space accurate)하다고 한다. 식(8.33)으로부터 명시적 유한차분법은 1차 시간 정확하고 2차 공간 정확함을 알 수 있다.

```
if (S[i]>=q)
q=S[i];
}
return q;
}
class heatex_convergence_test
{
private:
double m_T,m_h, m_k, m_alpha;
int m_Nt, m_start, m_end,m_N;
public:
heatex_convergence_test(double m_T, int m_start, int m_end);
~heatex_convergence_test(){};
void WriteData(vector<double> S, vector<vector<double>> CP);
void Convergence_Test();
};
heatex_convergence_test::heatex_convergence_test(double nT, int nstart,
int nend)
{
m_T=nT;m_start=nstart; m_end=nend;m_alpha=0.1;
}
void heatex_convergence_test::Convergence_Test()
{

vector<double> hh(5),tt(5),err(5);
for (int iter=0; iter<5;iter++)
{
m_N=10*(int)pow(2.0,iter)+1;
vector<double> x=linspace(m_start, m_end, m_N);
m_h=x[2]-x[1];
m_k=m_alpha*pow(m_h,2);
m_Nt=(int)floor(m_T/m_k+0.5);
vector<vector<double>> u(m_N, vector<double>(m_Nt+1));
vector<double> exact(m_N);
for (int i=0;i<m_N; i++)
{
```

있다.

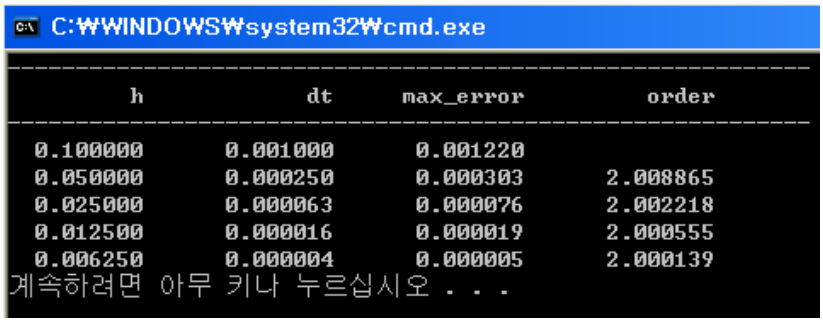


그림 8.8: 명시적 유한 차분법 수렴성 테스트

2.4.2 함축적 유한차분법

열방정식의 함축적 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자.

```
//////////////////////+heatim_convergence_test.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include <algorithm>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
```

```
{
vector<double> hh(5),tt(5),err(5);
for (int iter=0; iter<5;iter++)
{
m_N=10*(int)pow(2.0,iter)+1;
vector<double> x=linspace(m_start, m_end, m_N);
m_h=x[2]-x[1];
m_k=m_alpha*pow(m_h,2);
m_Nt=floor(m_T/m_k+0.5);
vector<vector<double>> u(m_N, vector<double>(m_Nt+1));
vector<double> exact(m_N);
for (int i=0; i<m_N;i++)
{
u[i][0]=sin(PI*x[i]);
exact[i]=u[i][0]*exp(-pow(PI,2)*m_T);
}
vector<double> dd(m_N-2),c(m_N-2),a(m_N-2);
for (int i=0; i<m_N-2;i++)
{
dd[i]=1+2.0*m_alpha;
c[i]=-m_alpha;
a[i]=-m_alpha;
}
vector<double> b(m_N-2),d(m_N-2);
for (int n=0;n<m_Nt;n++)
{
d=dd;
for (int i=0; i<m_N-2;i++)
b[i]=u[i+1][n];
for (int i=1; i<m_N-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
u[m_N-2][n+1]=b[m_N-3]/d[m_N-3];
```

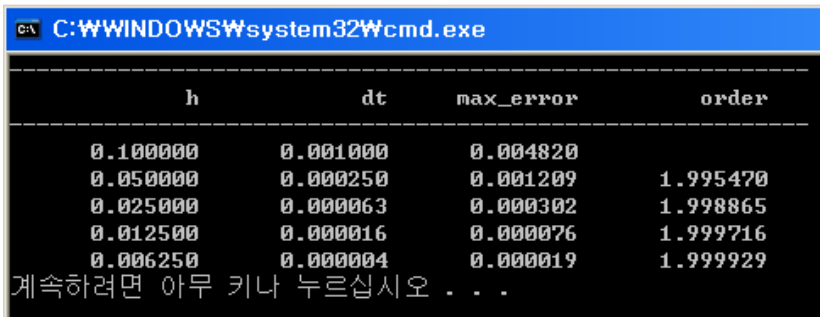


그림 8.9: 함축적 유한 차분법 수렴성 테스트

2.4.3 크랭크 니콜슨 유한차분법

열방정식의 크랭크 니콜슨 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자.

```
//////////////////////+heatcn_convergence_test.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include <algorithm>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
vector<double> absolute(vector<vector<double>> S, vector<double> SS, int
nFirst, int nSecond)
```

```
for (int iter=0; iter<5;iter++)
{
m_N=10*(int)pow(2.0,iter)+1;
vector<double> x=linspace(m_start, m_end, m_N);
m_h=x[2]-x[1];
m_k=m_alpha*pow(m_h,2);
m_Nt=(int)floor(m_T/m_k+0.5);
vector<vector<double>> u(m_N, vector<double>(m_Nt+1));
vector<double> exact(m_N);
for (int i=0; i<m_N;i++)
{
u[i][0]=sin(PI*x[i]);
exact[i]=u[i][0]*exp(-pow(PI,2)*m_T);
}
vector<double> dd(m_N-2),c(m_N-2),a(m_N-2);
for (int i=0; i<m_N-2;i++)
{
dd[i]=2*(1+m_alpha);
c[i]=-m_alpha;
a[i]=-m_alpha;
}
vector<double> b(m_N-2),d(m_N-2);
for (int n=0;n<m_Nt;n++)
{
d=dd;
for (int i=0; i<m_N-2;i++)
{
 b[i]=m_alpha*u[i][n]+2*(1-m_alpha)*u[i+1][n]+m_alpha*u[i+2][n];
}
for (int i=1; i<m_N-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
u[m_N-2][n+1]=b[m_N-3]/d[m_N-3];
```

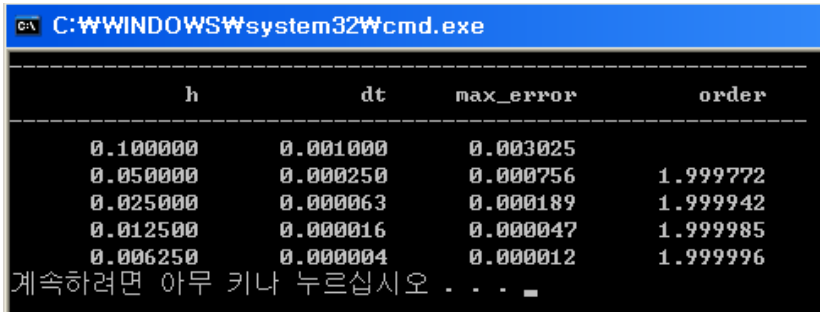


그림 8.10: 크랭크 니콜슨 유한 차분법 수렴성 테스트

제 3 절 Black-Scholes 편미분방정식에 대한 유한 차분법

유러피언 콜 옵션의 값을 구하기 위해서 Black-Scholes 편미분방정식을 유한차분법으로 풀어서 구한다. 편미분방정식은 Dirichlet 경계조건을 갖는 포물선형 편미분방정식이다. 특히 초기조건보다 만기시의 조건이 주어진다. $\tau = T - t$를 잔존기간으로 놓음으로써, 더 자연스러운 시간의 방정식으로 바꿀 수 있다. 그러면 편미분방정식은 다음과 같이 정리된다.

$$u_\tau = \frac{1}{2}\sigma^2 x^2 u_{xx} + rxu_x - ru.$$

이 때, 정의역은 $x \geq 0$ 이고, 시간의 범위는 $0 \leq \tau \leq T$, 초기값은 $u(x,0) = \max(x-E,0)$ [2] 이고, 경계조건은 $u(0,\tau) = 0$, 값이 큰 x에 대해서 $u(x,\tau) \approx x - Ee^{-r\tau}$를 갖는다. x를 $0 \leq x \leq L$의 범위로 두고 $h = L/(N_x - 1)$와 $k = T/N_t$의 간격을 갖는 유한 차분 격자를 사용함으로써, 이산 해 $u_i^n \approx u((i-1)h,(n-1)k) = u(x_i,t^n)$를 계산할 수 있다. 모든 $1 \leq n \leq N_t$에서 초기 데이터에 의해 지정된 값 $u_i^1 = \max(x_i - E, 0)$ for $1 \leq i \leq N_x$, 그리고 경계조건에 의해 지정된 경계값 $u_1^n = 0$와 $u_{N_x}^n = L - Ee^{-rt^n}$을 갖게 된다.

[2] 초기값 $u(x,0) = \max(x-E,0)$은 $x < E$일 때 $u(x,0) = 0$을 $x \geq E$일 때 $u(x,0) = x - E$을 의미한다.

```
{
private:
int  m_E, m_L,m_Nx, m_Nt ;
double m_sigma, m_r,m_T;

public:
BSex(int nE, int nL, double nsigma, double nr, double nT, int nNx, int
nNt)
{
m_E=nE; m_L=nL; m_Nx=nNx; m_Nt=nNt;
m_sigma=nsigma; m_r=nr; m_T=nT;
}
~BSex(){};
void GetOptionPrice(void);
void WriteData(vector<double> S, vector<vector<double>> CP);
};
void BSex::WriteData(vector<double> S, vector<vector<double>> CP)
{
ofstream output;
output.open("BSex_x_Data.dat");
output<<fixed<<setprecision(4);
for (unsigned int i=0; i < S.size(); i++)
output<<setw(12)<<showpoint<<S[i];
output.clear();
output.close();
output.open("BSex_u_Data.dat");
for(unsigned int n=0;n<CP[0].size();n++)
{
for (unsigned int i=0;i<S.size();i++)
{
output<<setw(12)<<showpoint<<CP[i][n];
}
output<<endl;
}
output.clear();
output.close();
```

```
clc; clf; clear;

x = load('BSex_x_Data.dat');
u = load('BSex_u_Data.dat')';
plot(x,u(:,1:200:1001),'ko-')
axis image; axis([0 800 0 600])
```

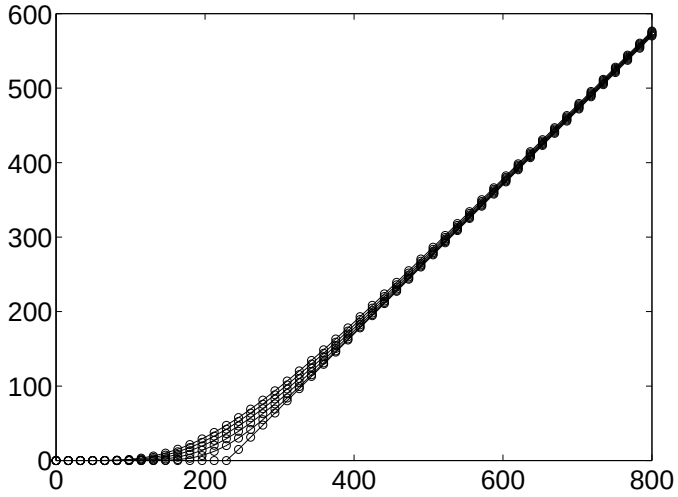


그림 8.11: 명시적 방법에 의한 블렉숄츠 방정식의 수치해

3.2 함축적 방법에 의한 옵션 가격 결정

시간에 대한 전방 차분을 이용한 명시적 방법에서 일어날 수 있는 불안정성 문제를 해결하기 위해서 후방차분을 이용하여 함축적 방법을 적용하면 다음의 식을 얻게 된다.

$$\begin{aligned} \frac{u_i^{n+1}-u_i^n}{k} &= \frac{1}{2}\sigma^2 x_i^2 \frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{h^2} \\ &\quad + rx_i \frac{u_{i+1}^{n+1}-u_{i-1}^{n+1}}{2h} - ru_i^{n+1}. \end{aligned} \tag{8.29}$$

이러한 함축적 방법은 시간의 크기에 영향을 받지 않을 뿐만 아니라 수치

```
BSim(double nE, int nL, double nsigma, double nr, double nT, int nNx,
int nNt)
{
m_E=nE; m_L=nL; m_sigma=nsigma; m_r=nr; m_T=nT; m_Nx=nNx; m_Nt=nNt;
}
~BSim(){};
void GetOptionPrice(void);
void WriteData(vector<double> S, vector<vector<double>> CP);
};
void BSim::WriteData(vector<double> S, vector<vector<double>> CP)
{
ofstream output;
output.open("BSim_x_Data.dat");
output<<fixed<<setprecision(4);
for (unsigned int i=0; i <S.size(); i++)
output<<setw(12)<<showpoint<<S[i];
output.clear();
output.close();
output.open("BSim_u_Data.dat");
for(unsigned int n=0;n<CP[0].size();n++)
{
for (unsigned int i=0;i<CP.size();i++)
{
output<<setw(12)<<showpoint<<CP[i][n];
}
output<<endl;
}
output.clear();
output.close();
}
void BSim::GetOptionPrice()
{
m_k=m_T/m_Nt;
vector<double> x=linspace(0, m_L, m_Nx);
m_h=x[1]-x[0];
vector<vector<double>> u(m_Nx, vector<double>(m_Nt+1));
```

```
int main()
{
BSim in(230.0, 800, 0.5, 0.03,1.0, 50, 100);
in.GetOptionPrice();
return 0;
}
```

다음은 data파일을 읽는 m-파일이다.

```
clc; clf; clear;

x = load('BSim_x_Data.dat');
u = load('BSim_u_Data.dat')';
plot(x,u(:,1:20:101),'ko-')
axis image; axis([0 800 0 600])
```

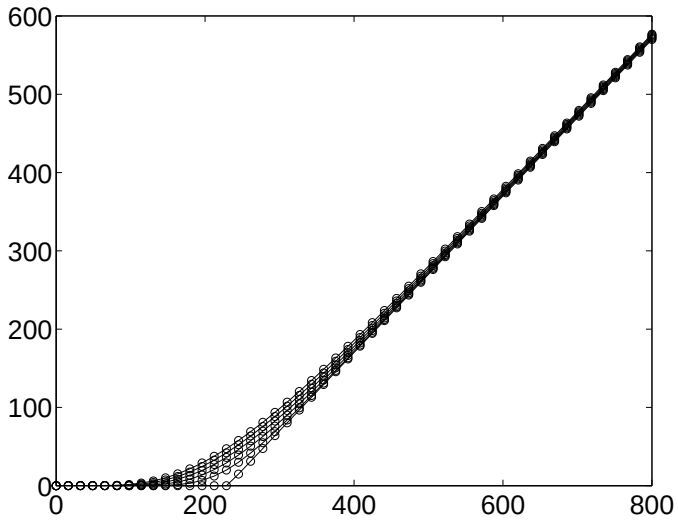


그림 8.12: 함축적 방법에 의한 옵션 가격 결정

```
using namespace std;
vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
class BScn
{
private:
double m_E, m_sigma, m_r, m_T, m_k, m_h;
int m_Nx, m_Nt, m_L ;
public:
BScn(double nE, int nL, double nsigma, double nr, double nT, int nNx,
int nNt)
{
m_E=nE; m_L=nL; m_sigma=nsigma; m_r=nr; m_T=nT; m_Nx=nNx; m_Nt=nNt;
}
~BScn(){};
void GetOptionPrice(void);
void WriteData(vector<double> S, vector<vector<double>> CP);
};
void BScn::WriteData(vector<double> S, vector<vector<double>> CP)
{
ofstream output;
output.open("BScn_x_Data.dat");
output<<fixed<<setprecision(4);
for (unsigned int i=0; i < S.size(); i++)
output<<setw(12)<<showpoint<<S[i];
output.clear();
output.close();
output.open("BScn_u_Data.dat");
for(unsigned int n=0;n<CP[0].size();n++)
{
```

```
u[m_Nx-1][n+1]=m_L-m_E*exp(-m_r*m_k*(n+1));
b[m_Nx-3]=b[m_Nx-3]-c[m_Nx-3]*u[m_Nx-1][n+1];
for (int i=1; i<m_Nx-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
u[m_Nx-2][n+1]=b[m_Nx-3]/d[m_Nx-3];
for (int i=m_Nx-4;i>=0; i--)
{
u[i+1][n+1]=(b[i]-c[i]*u[i+2][n+1])/d[i];
}
}
WriteData(x,u);
}
int main()
{
BScn cn(230.0, 800, 0.5, 0.03,1.0, 50, 100);
cn.GetOptionPrice();
return 0;
}
```

다음은 data파일을 읽는 m-파일이다.

```
clc; clf; clear;

x = load('BScn_x_Data.dat');
u = load('BScn_u_Data.dat')';
plot(x,u(:,1:20:101),'ko-')
```

```
a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}
vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
class BSex_stability
{
private:
double m_E, m_sigma, m_r, m_alpha,m_T, m_h, m_k;
int m_Nx, m_Nt,  m_L;
public:
BSex_stability(double nE, int nL, double nsigma, double nr, double nT,
int nNx)
{
m_E=nE; m_L=nL; m_sigma=nsigma; m_r=nr; m_alpha=0.000005; m_T=nT;
m_Nx=nNx;
}
~BSex_stability(){};
void GetOptionPrice(void);
void BSex_stability::WriteData(vector<double> S, vector<vector<double>>
CP,vector<vector<double>> CPP);
};
```

```
m_h=x[2]-x[1];
m_k=2*m_alpha*pow(m_h/m_sigma,2);
m_Nt=(int)floor(m_T/m_k+0.5);
vector<vector<double>> u(m_Nx, vector<double>(m_Nt+1)),exu(m_Nx,
vector<double>(m_Nt+1));
for(int i=0;i<m_Nx;i++)
{
if (x[i]<=m_E)
u[i][0]=0;
else
u[i][0]=x[i]-m_E;
}
for (int n=1;n<m_Nt+1;n++)
u[m_Nx-1][n]=m_L-m_E*exp(-m_r*m_k*n);
exu=u;
vector<double> d1(m_Nx),d2(m_Nx);
for (int n=0; n< m_Nt; n++)
{
for (int i=1; i<m_Nx-1;i++)
{
u[i][n+1] = u[i][n] + m_k*(((double)1/2)*pow(m_sigma, 2) * pow(x[i], 2)*
((u[i+1][n]-2.0*u[i][n]+u[i-1][n]) / pow(m_h,2)) +
m_r*x[i] * ((u[i+1][n]-u[i-1][n])/(2.0*m_h))-m_r*u[i][n]);
}

for(int i=0;i<m_Nx;i++)
{
d1[i]=(log(x[i]/m_E)+(m_r+pow(m_sigma,2)/2)*m_k*n) /
(m_sigma*sqrt(m_k*n));
d2[i]=d1[i]-m_sigma*sqrt(m_k*n);
exu[i][n+1]=x[i]*normcdf(d1[i])-m_E*exp(-m_r*m_k*n)*normcdf(d2[i]);
}
}
WriteData(x, u,exu);
}
int main()
```

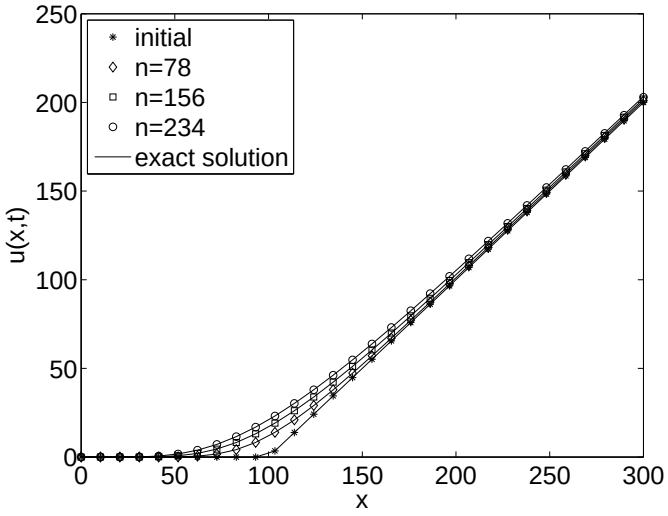


그림 8.14: $\Delta t = 0.0043$인 안정한 상태

```
//////////////////////+BSim_stability.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
double normcdf(double X)
{
double L, K, w ;
```

```
m_E=nE; m_L=nL; m_sigma=nsigma; m_r=nr; m_alpha=0.000005; m_T=nT;
m_Nx=nNx;
}
~BSim_stability(){};
void GetOptionPrice(void);
void WriteData(vector<double> S, vector<vector<double>>
CP,vector<vector<double>> CPP);
};
void BSim_stability::WriteData(vector<double> S, vector<vector<double>>
CP,vector<vector<double>> CPP)
{
ofstream output;
output.open("BSim_stability_x_Data.dat");
output<<fixed<<setprecision(4);
for (unsigned int i=0; i < S.size(); i++)
output<<setw(12)<<showpoint<<S[i];
output.clear();
output.close();
output.open("BSim_stability_u_Data.dat");
for(unsigned int n=0;n<CP[0].size();n++)
{
for (unsigned int i=0;i<CP.size();i++)
{
output<<setw(12)<<showpoint<<CP[i][n];
}
output<<endl;
}
output.clear();
output.close();
output.open("BSim_stability_exu_Data.dat");
for(unsigned int n=0;n<CPP[0].size();n++)
{
for (unsigned int i=0;i<CPP.size();i++)
{
output<<setw(12)<<showpoint<<CPP[i][n];
}
```

```
b[m_Nx-3]=u[m_Nx-2][n]/m_k-c[m_Nx-3]*u[m_Nx-1][n+1];
for (int i=1; i<m_Nx-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
u[m_Nx-2][n+1]=b[m_Nx-3]/d[m_Nx-3];
vector<double> d1(m_Nx),d2(m_Nx);

for (int i=m_Nx-4;i>=0; i--)
u[i+1][n+1]=(b[i]-c[i]*u[i+2][n+1])/d[i];
for(int i=0;i<m_Nx;i++)
{
d1[i]=(log(x[i]/m_E)+(m_r+pow(m_sigma,2)/2) * m_k*(n+1)) /
(m_sigma*sqrt(m_k*(n+1)));
d2[i]=d1[i]-m_sigma*sqrt(m_k*(n+1));
exu[i][n+1]=x[i]*normcdf(d1[i])-m_E*exp(-m_r*m_k*(n+1))*normcdf(d2[i]);
}
}
WriteData(x,u,exu);
}

int main()
{
BSim_stability im(100, 300, 0.5, 0.03, 1.0, 30);
im.GetOptionPrice();
return 0;
}
```

다음은 data파일을 읽는 m-파일이다.

```
clc; clf; clear;
Nt=234;L=300;E=100;
x = load('BSim_stability_x_Data.dat');
```

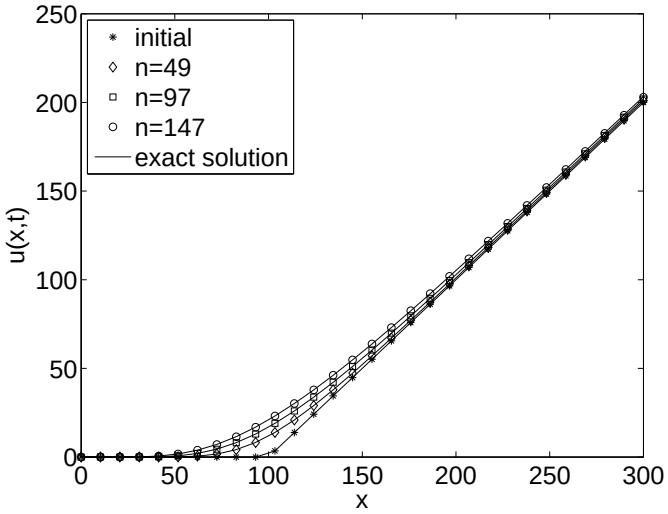


그림 8.17: $\Delta t = 0.0068$인 안정한 상태

3.5 수렴성 테스트

$u(x_i, t^n)$는 Black-Scholes 방정식의 정확한 해를 나타낸다. 다음은 정확한 해를 수치기법에 대입함으로써 명시적 유한차분법의 국소절단오차를 찾는 과정이다. 노드 (x_i, t^n)에서 국소절단오차는 다음과 같이 구한다.

$$\begin{aligned} T(x_i,t^n) &= \frac{u(x_i,t^{n+1})-u(x_i,t^n)}{k} \\ &\quad -\frac{\sigma^2 x_i^2}{2}\frac{u(x_{i+1},t^n)-2u(x_i,t^n)+u(x_{i-1},t^n)}{h^2} \\ &\quad -rx_i\frac{u(x_{i+1},t^n)-u(x_{i-1},t^n)}{2h}+ru(x_i,t^n). \end{aligned}$$

이제 노드(x_i, t^n)에서 테일러전개를 하면 각각의 항을 다음과 같이 나타낼 수 있다.

$$\begin{aligned} T(x_i,t^n) &= u_t(x_i,t^n)+\frac{k}{2}u_{tt}(x_i,t^n)+\mathcal{O}(k^2) \\ &\quad -\frac{\sigma^2 x_i^2}{2}\left[u_{xx}(x_i,t^n)+\frac{h^2}{12}u_{xxxx}(x_i,t^n)+\mathcal{O}(h^4)\right] \\ &\quad -rx_i\left[u_x(x_i,t^n)+\frac{h^2}{3}u_{xxx}(x_i,t^n)+\mathcal{O}(h^4)\right]+ru(x_i,t^n). \end{aligned}$$

```
return d;
}
double normcdf(double X)
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937,
a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}
vector<double> absolute(vector<vector<double>> S, vector<double> SS, int
nFirst, int nSecond)
{
vector<double> q;
for (int i=0; i < nFirst; i++) {
q.push_back(abs(S[i][nSecond-1] - SS[i]));
}
return q;
}
double sqrtsum(vector<double> S)
{
double q=0.0;
for (unsigned int i=0; i < S.size(); i++) {
q+=pow(S[i],2);
}
return q;
}

class BSex_convergence_test
{
```

```
{
u[i][n+1] = u[i][n] + m_k*(((double)1/2)*pow(m_sigma, 2) * pow(x[i], 2)*
((u[i+1][n]-2.0*u[i][n]+u[i-1][n]) / pow(m_h,2)) +
m_r*x[i]* ((u[i+1][n]-u[i-1][n])/(2.0*m_h))-m_r*u[i][n]);
}
}
vector<double> d1(m_Nx),d2(m_Nx),exact(m_Nx);
for (int i=0; i<m_Nx; i++)
{
d1[i]=(log(x[i]/m_E)+(m_r+pow(m_sigma,2)/2)*m_k*m_Nt) /
(m_sigma*sqrt(m_k*m_Nt));
d2[i]=d1[i]-m_sigma*sqrt(m_k*m_Nt);
exact[i]=x[i]*normcdf(d1[i])-m_E*exp(-m_r*m_k*m_Nt)*normcdf(d2[i]);
}

hh[iter]=m_h;
tt[iter]=m_k;
vector<double> F(m_Nx);
double G;
F=absolute(u,exact,m_Nx,m_Nt+1);
G=sqrtsum(F);
err[iter]=sqrt(G/m_Nx);

}
cout<<fixed<<setprecision(6);
cout << "------------------------------------------------------------" <<
endl;
cout <<setw(14)<<"h"<<setw(14)<<"dt"<<setw(14)<<"max_error"<<setw(14)
<<"order"<< endl;
cout << "------------------------------------------------------------" <<
endl;
cout <<setw(14)<<hh[0]<<setw(14)<<tt[0]<<setw(14)<<err[0]<<setw(14) <<
endl;
for (int i=1; i<5;i++)
{
```

형 콜옵션의 가격을 구하는 C++ 코드이다. 이 때, 초기 조건은 $u(x,0) = \max(0, x-E)$, $T = 0.1$, $L = 400$, $h = 1/N$, $\Delta t = h/500$을 이용하였다. C++ 코드BSex_convergence_test.cpp을 실행하면 다음의 결과를 얻을 수 있다.

```
//////////////////////+BSim_convergence_test.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
double normcdf(double X)
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937,
a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}
```

```
{
m_Nx=16*(int)pow(2.0,iter+1);
vector<double> x=linspace(0,m_L,m_Nx);
m_h=x[2]-x[1];
m_k=m_h/500;
m_Nt=floor(m_T/m_k+0.5);
vector<vector<double>> u(m_Nx,vector<double>(m_Nt+1));
for(int i=0;i<m_Nx;i++)
{
if (x[i]<=m_E)
u[i][0]=0;
else
u[i][0]=x[i]-m_E;
}

for (int n=1;n<m_Nt+1;n++)
u[m_Nx-1][n]=m_L-m_E*exp(-m_r*m_k*n);
vector<double> dd(m_Nx-2), c(m_Nx-2),a(m_Nx-2);
for (int i=0; i<m_Nx-2;i++)
{
dd[i]=1/m_k+pow(m_sigma*(i+1),2)+m_r;
c[i]=-m_r*(i+1)*0.5-pow(m_sigma*(i+1),2)*0.5;
a[i]=m_r*(i+2)*0.5-pow(m_sigma*(i+2),2)*0.5;
}
vector<double> b(m_Nx-2),d(m_Nx-2);
for (int n=0;n<m_Nt;n++)
{
d=dd;
for (int i=0; i<m_Nx-3;i++)
b[i]=u[i+1][n]/m_k;
b[m_Nx-3]=u[m_Nx-2][n]/m_k-c[m_Nx-3]*u[m_Nx-1][n+1];
for (int i=1; i<m_Nx-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
```

```
cout<<setw(10)<<hh[i]<<setw(14)<<tt[i]<<setw(14)<<err[i]<<setw(14) <<
log(err[i-1]/err[i])/log(2.0)<<setw(14)<<endl;
}
}

int main()
{
BSim_convergence_test im(100, 400, 0.5, 0.03, 0.1);
im.GetOptionPrice();
return 0;
}
```

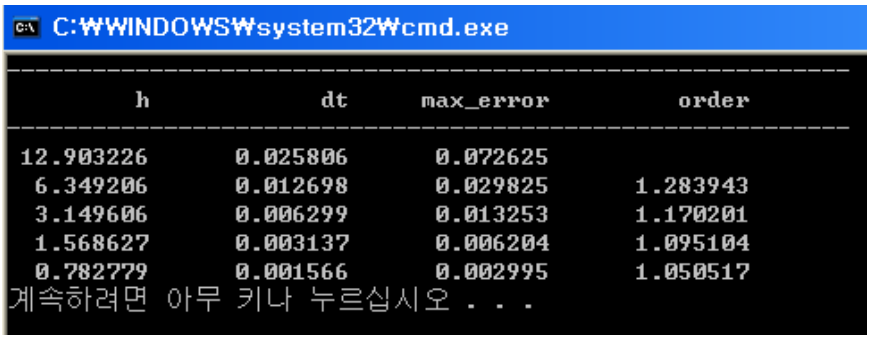


그림 8.19: 함축적 유한 차분법의 수렴성 테스트

위 결과로 부터 Black-Scholes 방정식에 대한 함축적 유한차분법의 수렴도가 1차임을 알수가 있다.

3.5.3 크랭크 니콜슨 유한차분법

유러피언 콜옵션방정식의 크랭크 니콜슨 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자. 무위험이자율이 $r = 0.03$, 변동성이 $\sigma = 0.5$, 현재시점이 $t = 0$, 만기 시점이 $T = 0.1$, 그리고 행사가격이 $E = 100$인 유럽형 콜옵션의 가격을 구하는 C++ 코드이다. 이 때, 초기 조건은 $u(x, 0) = \max(0, x - E)$, $T = 0.1$, $L = 400$, $h = 1/N$, $\Delta t = h/500$을 이용하였다. C++ 코드`BScn_convergence_test.cpp`을 실행하면 다음의 결과를 얻을 수 있다.

```
q.push_back(abs(S[i][nSecond-1] - SS[i]));
}
return q;
}
double sqrtsum(vector<double> S)
{
double q=0.0;
for (unsigned int i=0; i < S.size(); i++) {
q+=pow(S[i],2);
}
return q;
}

class BScn_convergence_test
{
private:
int  m_E, m_L,m_Nx, m_Nt ;
double m_sigma, m_r,m_T,m_h,m_k;

public:
BScn_convergence_test(int nE, int nL, double nsigma, double nr, double
nT) :
  m_E(nE), m_L(nL), m_sigma(nsigma), m_r(nr), m_T(nT)
  {}
  ~BScn_convergence_test(){};
  void GetOptionPrice(void);
};
void BScn_convergence_test::GetOptionPrice()
{
vector<double> hh(5),tt(5),err(5);
for (int iter=0; iter<5;iter++)
{
m_Nx=16*(int)pow(2.0,iter+1);
vector<double> x=linspace(0,m_L,m_Nx);
m_h=x[2]-x[1];
m_k=m_h/500;
```

```
u[i+1][n+1]=(b[i]-c[i]*u[i+2][n+1])/d[i];
}
}
vector<double> d1(m_Nx),d2(m_Nx),exact(m_Nx);
for (int i=0; i<m_Nx; i++)
{
d1[i]=(log(x[i]/m_E)+(m_r+pow(m_sigma,2)/2)*m_k*m_Nt) /
(m_sigma*sqrt(m_k*m_Nt));
d2[i]=d1[i]-m_sigma*sqrt(m_k*m_Nt);
exact[i]=x[i]*normcdf(d1[i])-m_E*exp(-m_r*m_k*m_Nt)*normcdf(d2[i]);
}
hh[iter]=m_h;
tt[iter]=m_k;
vector<double> F(m_Nx);
double G;
F=absolute(u,exact,m_Nx,m_Nt+1);
G=sqrtsum(F);
err[iter]=sqrt(G/m_Nx);
}
cout<<fixed<<setprecision(6);
cout << "-------------------------------------------------------------" <<
endl;
cout <<setw(10)<<"h"<<setw(14)<<"dt"<<setw(14)<<"max_error"<<setw(14) <<
"order"<< endl;
cout << "-------------------------------------------------------------" <<
endl;
cout <<setw(10)<<hh[0]<<setw(14)<<tt[0]<<setw(14)<<err[0]<<setw(14) <<
endl;
for (int i=1; i<5;i++)
{
cout<<setw(10)<<hh[i]<<setw(14)<<tt[i]<<setw(14)<<err[i]<<setw(14) <<
log(err[i-1]/err[i])/log(2.0)<<setw(14)<<endl;
}
}

int main()
```

4.1 Greeks

블랙숄즈의 옵션가격결정모형에서 콜옵션의 가치 V는 다음과 같다.

$$\begin{aligned} V &= SN(d_1) - Ee^{-rT}N(d_2) \\ d_1 &= \frac{\ln(\frac{S}{E}) + \left(r + \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}} \\ d_2 &= \frac{\ln(\frac{S}{E}) + \left(r - \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T} \end{aligned} \tag{8.34}$$

표준정규누적분포함수와 미분식은 다음과 같이 정의된다.

$$\begin{aligned} N(d) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d} e^{-\frac{x^2}{2}}dx \\ N'(d) &= \frac{1}{\sqrt{2\pi}}e^{-\frac{d^2}{2}} \end{aligned}$$

Greeks	
Delta (Δ)	$\frac{\partial V}{\partial S} = N(d_1)$
Gamma (Γ)	$\frac{\partial^2 V}{\partial S^2} = N'(d_1)/(S\sigma\sqrt{T})$
Theta (Θ)	$\frac{\partial V}{\partial t} = -S\sigma N'(d_1)/(2\sqrt{T}) - rEe^{-rT}N(d_2)$
Rho (ρ)	$\frac{\partial V}{\partial r} = ETe^{-rT}N(d_2)$
Vega	$\frac{\partial V}{\partial \sigma} = S\sqrt{T}N'(d_1)$

Greeks는 말 그대로 그리스 문자들을 말한다. 각각의 그리스 문자는 각각의 지표의 변동에 대한 옵션가치의 변동비율, 즉 민감도를 의미한다. 이들 그리스문자들은 헤징과 관련하여 옵션의 투자 전략에 매우 중요하게 사용된다. Greeks은 옵션의 가격에 대한 해석해가 있는 경우, 각각의 Greeks의 정의에 따라 구한다. 그러나 대다수의 이색옵션과 같이 옵션가격의 해석해를 구할 수 없는 경우에는 수치 기법에 의존하여 Greeks의 근사적인 값을 구해야 한다.

폴리오를 복제포트폴리오라고 한다. 풋옵션의 델타는 항상 음의 값을 가진다. 따라서 무위험 포트폴리오를 구성하기 위해, 풋옵션의 포지션과 기초자산의 포지션은 동일하게 설정한다.

Greeks의 수치해를 구할때 편의상 함축적 유한차분법을 사용할것이나 좀 더 정확한 해를 구하기 위해서 크랭크-니콜슨 방법을 사용할 수도 있다. `BSim_delta.cpp`는 수치해와 블랙-숄즈 해석해를 이용해 델타를 구하는 C++ 코드이다.

```
//////////////////////+BSim_delta.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)
vector<double> linspace(int START,int  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
double normcdf(double X)
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937,
a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
```

```
output.open("BSim_delta_uu_Data.dat");
for (unsigned int i=0;i<CPP.size();i++)
output<<setw(12)<<showpoint<<CPP[i];
output.clear();
output.close();
}

void BSim_delta::Get_Delta()
{
m_k=m_T/m_Nt;
vector<double> x=linspace(0,m_L, m_Nx);
m_h=x[1]-x[0];
vector<double> v;
for(int i=0; i<m_Nx;i++)
{
if (x[i]<=m_E)
v.push_back(0);
else
v.push_back(x[i]-m_E);
}
vector<double> dd(m_Nx-2),c(m_Nx-2),a(m_Nx-3),d(m_Nx),b(m_Nx);
for(int i=0;i<m_Nx-2;i++)
{
dd[i]=1/m_k+pow(m_sigma*(i+1),2)+m_r;
c[i]=-m_r*(i+1)/2-pow(m_sigma*(i+1),2)/2;
}
for(int i=0;i<m_Nx-3;i++)
a[i]=m_r*(i+2)/2-pow(m_sigma*(i+2),2)/2;
for(int n=0;n<m_Nt;n++)
{
d=dd;
for(int i=0;i<m_Nx-3;i++)
b[i]=v[i+1]/m_k;
v[m_Nx-1]=m_L-m_E*exp(-m_r*m_k*n);
b[m_Nx-3]=v[m_Nx-2]/m_k-c[m_Nx-3]*v[m_Nx-1];
for(int i=1;i<m_Nx-2;i++)
```

```
Delta2 = load('BSim_delta_uu_Data.dat');
hold on
plot(x(2:Nx-1),Delta,'k*-');
plot(x,Delta2,'ko-'); grid on
hold off
xlabel('Underlying Asset','fontsize',20)
ylabel('Delta','fontsize',20)
legend('FDM','Exact',4)
set(gca,'fontsize',20)
```

그림 8.21은 C++ 코드 BSim_delta.cpp를 실행한 결과이다. 수치해석에 의한 델타와 해석해에 의한 델타가 매우 비슷함을 알 수 있다.

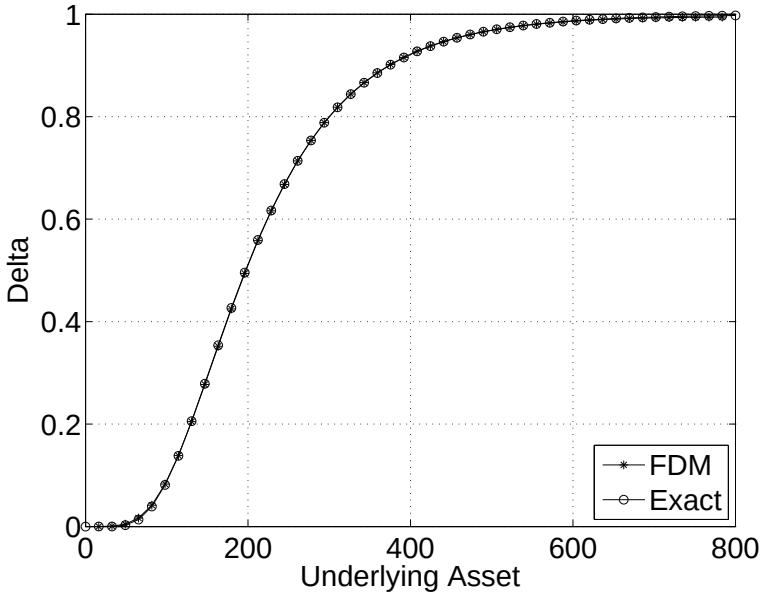


그림 8.21: 유한차분법과 해석해을 이용하여 구한 델타

```
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937
, a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}

class BSim_gamma
{
private:
int  m_E, m_L,m_Nx, m_Nt ;
double m_sigma, m_r,m_T,m_h,m_k;

public:
BSim_gamma(int nE, int nL, double nsigma, double nr, double nT, int nNx,
int nNt) :
  m_E(nE), m_L(nL), m_sigma(nsigma), m_r(nr), m_T(nT), m_Nx(nNx),
m_Nt(nNt)
  {}
  ~BSim_gamma(){};
  void WriteData(vector<double> S, vector<double> CP,vector<double>
CPP);
  void Get_gamma(void);
};
void BSim_gamma::WriteData(vector<double> S, vector<double>
CP,vector<double> CPP)
{
ofstream output;
output.open("BSim_gamma_x_Data.dat");
```

```
a.push_back(m_r*(i+2)/2-pow(m_sigma*(i+2),2)/2);
for(int n=0;n<m_Nt;n++)
{
d=dd;
for(int i=0;i<m_Nx-3;i++)
b[i]=v[i+1]/m_k;
v[m_Nx-1]=m_L-m_E*exp(-m_r*m_k*(n+1));
b[m_Nx-3]=v[m_Nx-2]/m_k-c[m_Nx-3]*v[m_Nx-1];
for(int i=1;i<m_Nx-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
v[m_Nx-2]=b[m_Nx-3]/d[m_Nx-3];
for(int i=m_Nx-4;i>=0;i--)
{
v[i+1]=(b[i]-c[i]*v[i+2])/d[i];
}
}
vector<double> Gamma1,Gamma2;
Gamma1.push_back(0);
for(int i=1; i<m_Nx-1;i++)
Gamma1.push_back((v[i+1]-2*v[i]+v[i-1])/pow(m_h,2));
for(int i=0; i<m_Nx;i++)
{
double d=(log(x[i]/m_E)+(m_r+pow(m_sigma,2)/2)*m_T)/(m_sigma*sqrt(m_T));
Gamma2.push_back((exp(-0.5*pow(d,2)))/(x[i]*m_sigma*sqrt(2*PI*m_T)));
}
Gamma2[0]=0;
WriteData(x,Gamma1,Gamma2);
}
int main()
{
BSim_gamma gamma(230, 800,0.5,0.03,1.0,50, 100);
gamma.Get_gamma();
```

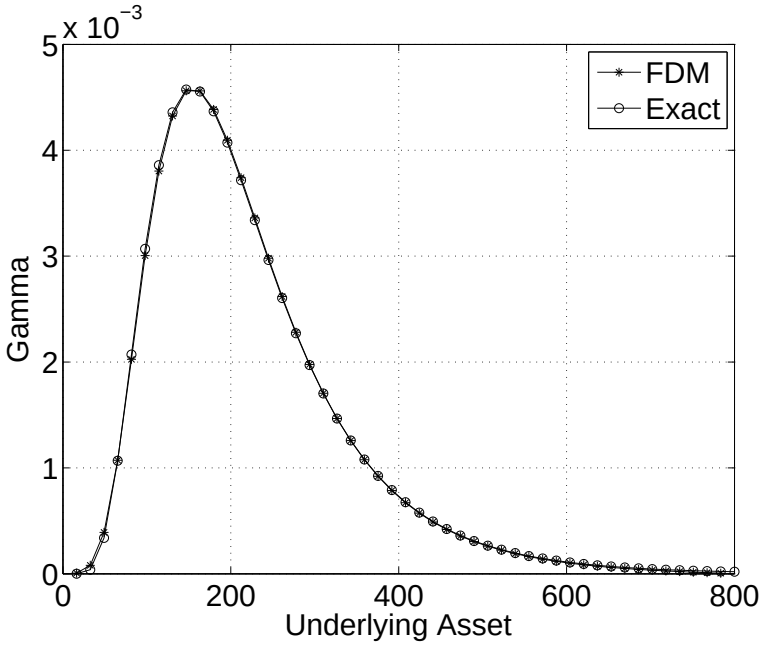


그림 8.22: 유한차분법을 이용하여 구한 감마

```
#include<vector>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

double normcdf(double X)
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937
, a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
```

```
output.open("BSim_theta_x_Data.dat");
output<<fixed<<setprecision(4);
for (unsigned int i=0; i < S.size(); i++)
output<<setw(14)<<showpoint<<S[i];
output.clear();
output.close();
output.open("BSim_theta_u_Data.dat");
for (unsigned int i=0;i<CP.size();i++)
output<<setw(14)<<showpoint<<CP[i];
output.clear();
output.close();
output.open("BSim_theta_uu_Data.dat");
for (unsigned int i=0;i<CPP.size();i++)
output<<setw(14)<<showpoint<<CPP[i];
output.clear();
output.close();
}
void BSim_theta::Get_theta()
{
m_k=m_T/m_Nt;
vector<double> x=linspace(0,m_L, m_Nx);
m_h=x[1]-x[0];
vector<double> v,ov;
for(int i=0; i<m_Nx;i++)
{
if (x[i]<=m_E)
v.push_back(0);
else
v.push_back(x[i]-m_E);
}
vector<double> dd(m_Nx-2),c(m_Nx-2),a(m_Nx-3),d(m_Nx),b(m_Nx);
for(int i=0;i<m_Nx-2;i++)
{
dd[i]=1/m_k+pow(m_sigma*(i+1),2)+m_r;
c[i]=-m_r*(i+1)/2-pow(m_sigma*(i+1),2)/2;
}
```

```
}
WriteData(x,theta1,theta2);
}
int main()
{
BSim_theta theta(230, 800,0.5,0.03,1.0,50, 100);
theta.Get_theta();
}
```

다음은 data 파일을 읽는 m-파일이다.

```
clc; clf; clear;
x = load('BSim_theta_x_Data.dat');
theta1 = load('BSim_theta_u_Data.dat');
theta2 = load('BSim_theta_uu_Data.dat');
hold on
plot(x, theta1,'k*-')
plot(x, theta2,'ko-');
hold off
grid on
xlabel('Underlying Asset','fontsize',20)
ylabel('Theta','fontsize',20)
legend('FDM','Exact',4)
set(gca,'fontsize',20)
```

그림 8.23은 C++ 코드를 실행한 결과이다. 유한차분법에 의한 세타와 해석해에 의한 세타는 비슷함을 알 수 있으며 행사가격 근처에서 급변함을 확인할 수 있다.

4.1.4 로우(ρ)

로우(ρ)는 무위험이자율 r의 변화에 따른 옵션가격의 변화율을 의미한다. 로우는 옵션의 가치 V를 무위험이자율 r에 대해 편미분해서 얻는다.

```
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
double normcdf(double X)
{
double L, K, w ;
double const a1 = 0.31938153, a2 = -0.356563782, a3 = 1.781477937
, a4 = -1.821255978, a5 = 1.330274429;
L = fabs(X);
K = 1.0 / (1.0 + 0.2316419 * L);
w = 1.0 - 1.0 / sqrt(2 * PI) * exp(-L *L / 2) *
(a1 * K + a2 * K *K + a3 * pow(K,3) + a4 * pow(K,4) + a5 * pow(K,5));
if (X < 0 ){
w= 1.0 - w;
}
return w;
}

class BSim_rho
{
private:
int  m_E, m_L,m_Nx, m_Nt ;
double m_sigma,m_T,m_h,m_k;
vector<double> m_r;

public:
BSim_rho(int nE, int nL, double nsigma, vector<double>nr, double nT, int
nNx, int nNt) :
  m_E(nE), m_L(nL), m_sigma(nsigma), m_T(nT), m_Nx(nNx),m_r(nr),m_Nt(nNt)
  {}
  ~BSim_rho(){};
```

```
{
if (x[i]<=m_E)
v[i]=0;
else
v[i]=x[i]-m_E;
}
vector<double> dd(m_Nx-2),c(m_Nx-2),a(m_Nx-3),d(m_Nx),b(m_Nx);
for(int i=0;i<m_Nx-2;i++)
{
dd[i]=1/m_k+pow(m_sigma*(i+1),2)+m_r[j];
c[i]=-m_r[j]*(i+1)/2-pow(m_sigma*(i+1),2)/2;
}
for(int i=0;i<m_Nx-3;i++)
a[i]=m_r[j]*(i+2)/2-pow(m_sigma*(i+2),2)/2;
for(int n=0;n<m_Nt;n++)
{
d=dd;
for(int i=0;i<m_Nx-3;i++)
b[i]=v[i+1]/m_k;
v[m_Nx-1]=m_L-m_E*exp(-m_r[j]*m_k*n);
b[m_Nx-3]=v[m_Nx-2]/m_k-c[m_Nx-3]*v[m_Nx-1];
for(int i=1;i<m_Nx-2;i++)
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
v[m_Nx-2]=b[m_Nx-3]/d[m_Nx-3];
for(int i=m_Nx-4;i>=0;i--)
{
v[i+1]=(b[i]-c[i]*v[i+2])/d[i];
}
}
if (j==0)
v0=v;
else
```

```
plot(x,Rho,'k*-'); hold on
plot(x,Rho2,'ko-'); grid on

xlabel('Underlying Asset','fontsize',20)
ylabel('Rho','fontsize',20)
legend('FDM','Exact',4)
set(gca,'fontsize',20)
```

그림 8.24은 C++ 코드를 실행한 결과이다. 유한차분법에 의한 로우와 해석해에 의한 로우는 비슷함을 알 수 있으며 행사가격 근처에서 급변함을 확인할 수 있다.

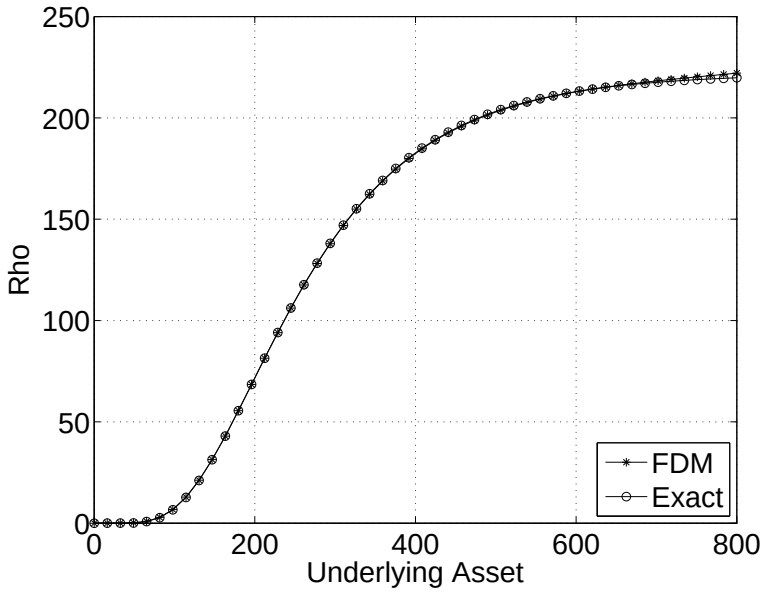


그림 8.24: 함축적 유한차분법을 이용하여 구한 로우

4.1.5 베가($Vega$)

베가는 변동성 σ에 대한 옵션가격의 변화율을 의미한다. 베가는 그리스문

```
}
return w;
}
vector<double> max(vector<double> S,double E)
{
vector<double> u;
for(unsigned int i=0;i<S.size();i++)
u.push_back(max(S[i]-E,0.0));
return u;
}
class BSim_vega
{
private:
int  m_E, m_L,m_Nx, m_Nt ;
double m_r,m_T,m_h,m_k;
vector<double> m_sigma;
public:
BSim_vega(int nE, int nL, vector<double> nsigma, double nr, double nT,
int nNx, int nNt) :
  m_E(nE), m_L(nL), m_sigma(nsigma), m_T(nT), m_Nx(nNx),m_r(nr),m_Nt(nNt)
  {}
  ~BSim_vega(){};
  void WriteData(vector<double> S, vector<double> CP,vector<double>
CPP);
  void Get_vega(void);
};
void BSim_vega::WriteData(vector<double> S, vector<double>
CP,vector<double> CPP)
{
ofstream output;
output.open("BSim_vega_x_Data.dat");
output<<fixed<<setprecision(4);
for (unsigned int i=0; i < S.size(); i++)
output<<setw(14)<<showpoint<<S[i];
output.clear();
output.close();
```

```
{
double xmult=a[i-1]/d[i-1];
d[i]=d[i]-xmult*c[i-1];
b[i]=b[i]-xmult*b[i-1];
}
v[m_Nx-2]=b[m_Nx-3]/d[m_Nx-3];
for(int i=m_Nx-4;i>=0;i--)
{
v[i+1]=(b[i]-c[i]*v[i+2])/d[i];
}
}
if (j==0)
v0=v;
else
v1=v;
}
vector<double> vega1;
for(int i=0; i<m_Nx;i++) {
vega1.push_back((v1[i]-v0[i])/(m_sigma[1]-m_sigma[0]));
}
vector<double> vega2;
for(int i=0; i<m_Nx;i++)
{
double sigma=(m_sigma[1]+m_sigma[0])/2;
double d1=(log(x[i]/m_E)+(m_r+pow(sigma,2)/2)*m_T)/(sigma*sqrt(m_T));
vega2.push_back(x[i]*sqrt(m_T)*exp(-0.5*pow(d1,2))/sqrt(2*PI));
}
WriteData(x,vega1,vega2);
}

int main()
{
vector<double> vsigma;
vsigma.push_back(0.4);
vsigma.push_back(0.5);
BSim_vega vega(230,800,vsigma,0.03,1.0,50,100);
```

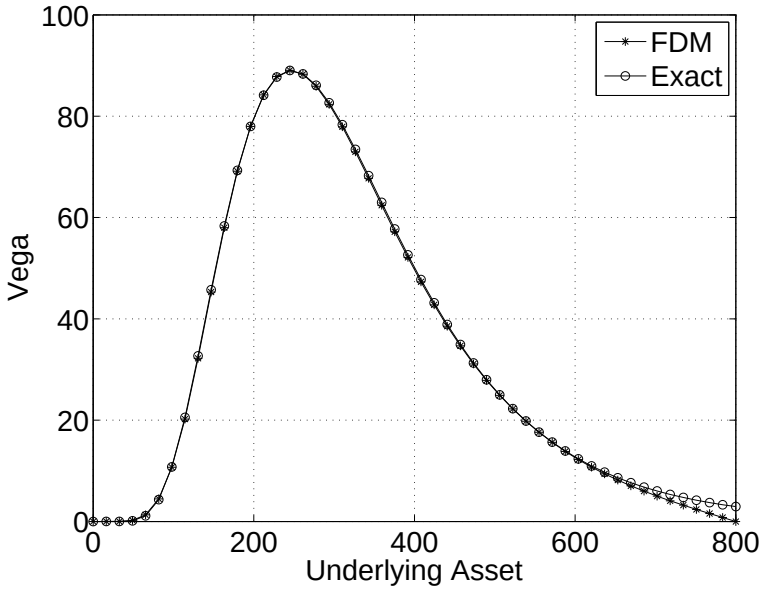


그림 8.25: 함축적 유한차분법을 이용하여 구한 베가

제 9 장

Tree가격결정모형 (Tree Pricing Model)

블랙-숄즈의 옵션가격결정모형은 기초자산의 가격변화가 연속적인 값을 갖는다고 가정하여 옵션의 가격을 도출한다. 이 모형을 풀기 위해서는 확률해석학과 편미분방정식 등을 비롯한 상당한 수학적 지식이 요구된다. 이에 비해 이산모형은 기초자산의 가격변화가 이산적인 값을 갖는다고 가정하고 옵션의 가격을 도출한다. 이 중에서 Tree 모형은 기초자산의 가격이 상승하거나 하락하는 두 가지 값만을 갖는 상황에서 옵션의 가치를 구한다. Tree 모형은 블랙-숄즈의 모형에 비해 수학적으로 더 쉽게 접근할 수 있다. 그 중에서 이항모형은 1979년에 Cox, Ross 그리고 Rubinstein이 옵션의 가격을 결정하기 위해 개발한 모형으로 위험중립가치평가원리를 적용하여 옵션의 가격을 산정한다.

제 1 절 이항옵션가격결정모형의 가정

이항옵션가격결정모형(이후 이항모형)은 다음과 같은 가정들로부터 도출된다.

(1) 완전자본시장가정: 자본시장에서 차익거래의 기회가 존재하지 않으며,

수 S가 기말에 Su가 되거나 Sd가 되는지의 여부에 따라 기초 자산의 옵션 가치 f와 기말의 옵션 가치 f_u와 f_d는 그림 9.2와 같이 나타낼 수 있다.

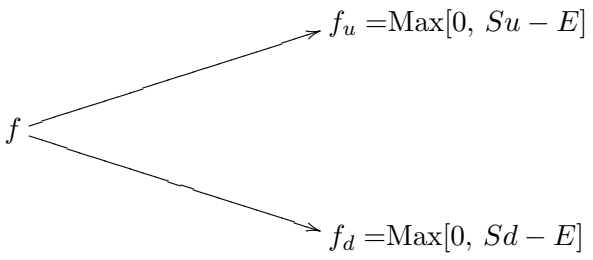


그림 9.2: 1기간 이항모형-옵션

블랙-숄즈모형에서 이용한 Covered call option strategy 즉, 콜옵션매도와 주식매수를 적절히 조합하면 무위험포트폴리오를 구성할 수 있다는 논리를 그대로 이용하여 콜옵션을 1개 매도하고 주식을 Δ개 매입하는 무위험 포트폴리오를 구성해보자. 투자자는 콜옵션매도로부터 받은 옵션프리미엄과 은행대출로 받은 금액으로 주식을 구입한다. 그림 9.3는 무위험 포트폴리오의 가치를 나타낸 것이다. 무위험 포트폴리오는 기초자산의 가격이

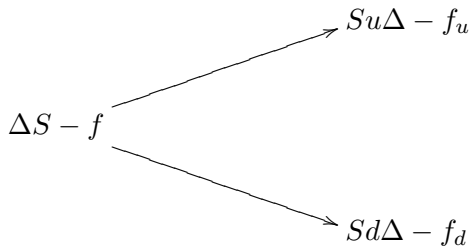


그림 9.3: 1기간 이항모형의 포트폴리오의 가치

오르거나, 내리거나 동일한 수익을 보장해주어야 하므로 아래의 식이 성립한다.

$$Su\Delta - f_u = Sd\Delta - f_d$$

Δ를 구해보면

$$\Delta = \frac{f_u - f_d}{Su - Sd} \left(\approx \frac{\partial f}{\partial S} \right) \tag{9.1}$$

를 확률이라고 하면, 이는 risk-neutral probability measure로 시장의 의견이 무시된 값이므로, Q는 p와 서로 다른 개념인 것이다. 그렇다면, Q가 왜 risk-neutral probability measure인지 알아보자.

위 식을 정리하면 옵션의 가치 f는

$$f = e^{-rT}(Qf_u + (1-Q)f_d)$$

이다. 기말주가 S_T의 기대값을 확률측도 Q을 이용하여 도출해 보자.

$$\begin{aligned} E(S_T) &= QSu + (1-Q)Sd = QS(u-d) + Sd \\ &= \frac{e^{rT}-d}{u-d}S(u-d) + Sd = Se^{rT} \end{aligned}$$

위의 식은 확률측도 Q를 사용할 경우 주가는 무위험이자율만큼의 비율로 증가해간다는 것을 보여준다. 즉, 확률측도 Q를 이용하여 주가를 계산하는 것이 바로 위험중립을 가정하는 것임을 보여준다. 위에서 계산한 옵션의 가치 f도 옵션의 가치가 위험중립가정하에서 미래의 기대수익을 무위험이자율로 할인한 현재가치이다. 따라서, Q를 위험중립확률(Risk Neutral Probability)이라고 부른다.

예제

현재 KOSPI200지수가 100이고, 1기간 후의 KOSPI200지수가 130이나 80이 된다고 가정해보자. 1기간 후에 배당이 없는 KOSPI200을 100에 매입할 수 있는 유럽형 콜옵션의 가격을 구해보자.

먼저, 문제를 다음과 같은 그림으로 표현해보자.
이러한 KOSPI200을 기초자산으로 하는 콜옵션의 이론가격 f를 구하기 위하여 콜옵션 1계약을 매도하고 Δ만큼 주식을 매입하여 무위험 포트폴리오를 만들어보자.
위의 무위험 포트폴리오는 KOSPI200지수의 상승 또는 하락에 관계없이 일정한 가치를 갖게 되므로 KOSPI200지수의 두 가지 움직임에 상관없이 포

션의 가격 f가 다음과 같이 구해진다.

$$100 \times \frac{3}{5} - f = 43.4322$$
$$f = 16.5678$$

따라서, 적정한 콜옵션의 가격 f는 16.5678이 된다.

이제, 위의 예제를 C++ 으로 구현해보자.

```
//////////////////////+binomial_1time.cpp+/////////////////////
#include<iostream>
#include<cmath>
#include<vector>
using namespace std;

vector<double> max(vector<double> S,double E)
{
vector<double> u;
for(unsigned int i=0;i<S.size();i++)
u.push_back(max(S[i]-E,0.0));
return u;
}
class binomial_1time
{
private:
int  m_S,m_E;
double m_r,m_T;
vector<double> m_St;

public:
binomial_1time(int nE, int nS,double nr, double nT,vector<double> nSt)
:
  m_E(nE), m_S(nS),m_r(nr), m_T(nT),m_St(nSt)
  {}
  ~binomial_1time(){};
  void Get_Callprice(void);

};
```

중립적 가치평가라는 것은 만기의 옵션가치의 기대값 $pf_u + (1-p)f_d$을 위험중립적 시장에서 기대현금흐름을 무위험이자율로 할인함을 의미하는 것이다.

예제

앞의 예제를 $S = 100, E = 100, u = 1.3, d = 0.8, r = 10\%$ 일때, 위험중립적 가치평가방식에 따라 다시 풀어보기로 하자.

먼저 KOSPI200지수의 상승확률 p를 구해보자.

$$p = \frac{e^{0.1} - 0.8}{1.3 - 0.8} = 0.6103$$

따라서, 기간 말에 옵션의 가치가 30이 될 확률을 0.6103이고, 0이 될 확률은 0.3897이 된다. 그러므로 옵션의 기대가치는

$$0.6103 \times 30 + 0.3897 \times 0 = 18.3103$$

이 되고, 이를 무위험이자율 10%로 할인하면 다음의 옵션가격을 얻을 수 있다.

$$18.3103 \times e^{-0.1} = 16.5678$$

이제, 위의 예제를 C++ 으로 구현해보자.

```
//////////////////////+binomial_1time2.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
using namespace std;

class binomial_1time2
{
private:
int  m_S,m_E,m_N;
double m_r,m_T,m_u,m_d,m_p;
```

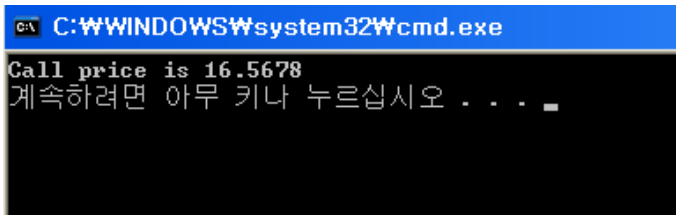


그림 9.5: 1기간 이항 모형-1

제 3 절 다기간 모형

앞 절에서 살펴본 1기간 이항모형은 다기간 이항모형으로 확장할 수 있다. 먼저 2기간 이항모형을 살펴보자.

3.1 2기간 모형

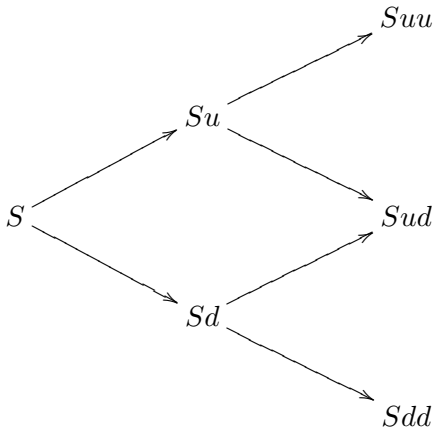


그림 9.6: 2기간 이항모형-주식

현재의 주가지수는 S이며 1기간마다 주가지수는 그 전기의 주가지수에 u를 곱한 만큼 상승하거나, d를 곱한 만큼 하락한다거 가정하자. 그러므로 1기간이 지난 경우 주가지수는 Su로 상승하거나 Sd로 하락한다고 가정하면 1기간이 경과한 후 주가지수는 Su와 Sd에 각각 u를 곱한 만큼 상승하거나 d를 곱한 만큼 하락하게 된다. 2기에서 주가지수는 Suu, Sud 또는

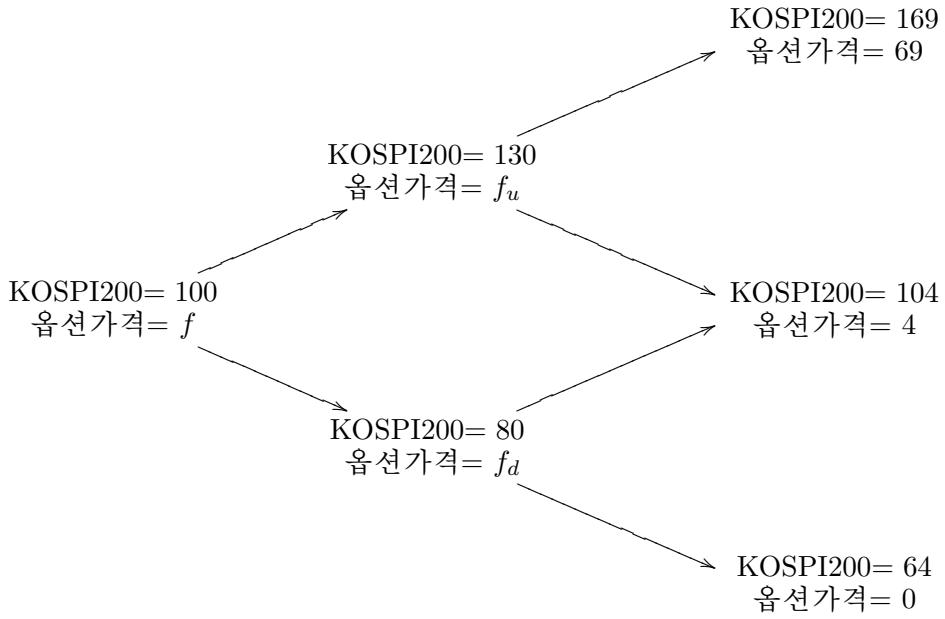
을 얻게된다.

예제

앞의 예제를 2기간 이항모형으로 확장시켜 생각해보자.

현재 100인 KOSPI200지수는 각 기간마다 30% 상승 또는 20% 하락한다고 가정하자. 2기간 후 KOSPI200지수를 100에 매입할 수 있는 유럽형 콜옵션의 가격을 구해보자.

2기간 동안 KOSPI200지수와 옵션가격을 아래의 그림과 같이 표현할 수 있다.



각 기간마다 30%의 상승비율 또는 20%의 하락비율을 가지고 있으므로, KOSPI200지수는 1기간 말에 130 또는 80이 되고, 2기간 말에는 169, 104, 64이 될 것이다. 위험 중립적 가치평가모형을 적용하기 위해 각 기간별 KOSPI200지

```
return u;
}

class binomial_2time
{
private:
int  m_S,m_E,m_N;
double m_r,m_T,m_p,m_dt,m_u,m_d;
public:
binomial_2time(int nE, int nS,int nN,double nr, double nT,double
nu,double nd)  :
  m_E(nE), m_S(nS),m_r(nr), m_T(nT),m_u(nu),m_d(nd),m_N(2)
  {}
  ~binomial_2time(){};
  void Get_Callprice(void);
};
void binomial_2time::Get_Callprice()
{
m_dt=m_T/m_N;
m_p=(exp(m_r*m_dt)-m_d)/(m_u-m_d);
vector<vector<double>> m_St(m_N+1,vector<double>(m_N+1));
m_St[0][0]=m_S;
for(int j=1;j<m_N+1;j++)
{
for(int i=0;i<=j;i++)
{
m_St[i][j]=m_S*pow(m_u,(j-i))*pow(m_d,i);
}
}
vector<vector<double>> call(m_N+1,vector<double>(m_N+1));
for (int i=0;i<m_N+1;i++)
{
call[i][m_N]=max(m_St[i][m_N]-m_E,0.0);
}
for(int j=m_N-1;j>=0;j--)
{
```

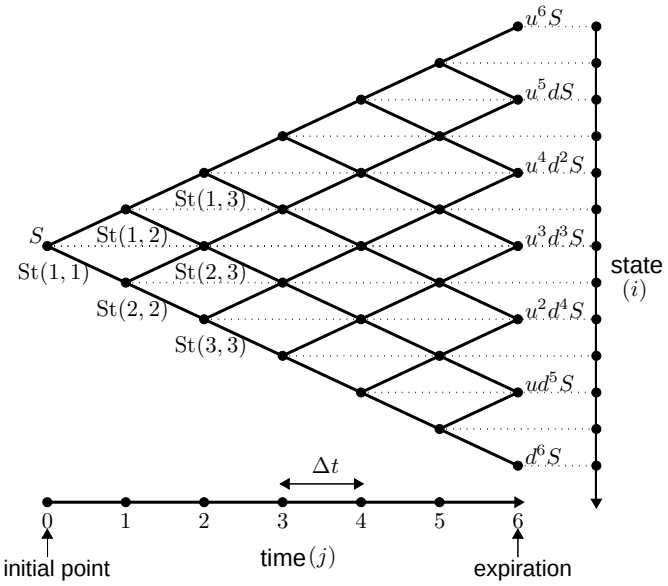


그림 9.9: 이항옵션

따라서 Δt기간말의 주가의 기대값은 $Se^{r\Delta t}$이므로 다음의 식이 성립한다.

$$\begin{aligned} Se^{r\Delta t} &= pSu + (1-p)Sd \\ e^{r\Delta t} &= pu + (1-p)d \end{aligned}$$

주가가 랜덤워크를 따른다는 가정하에 아주 짧은 시간 Δt동안 주가변화율의 분산은 $\sigma^2\Delta t$이다. 그러면 분산의 정의에 따라,

$$\sigma^2\Delta t = pu^2 + (1-p)d^2 - [pu + (1-p)d]^2$$

가 성립한다. 위의 공식은 간단한 대수적 조작을 거쳐 아래의 식이 된다.

$$\sigma^2\Delta t = e^{r\Delta t}(u+d) - ud - e^{2r\Delta t} \tag{9.6}$$

여기에 Cox와 Ross, Rubinstein은 다음과 같은 가정을 제시했다.

$$u = \frac{1}{d}$$

$u = 1/d$이므로 $Sud = S$가 된다. 이는 주가가 상승한 후에 하락한 가격은 초기의 주가와 같다는 것이다. 이 가정은 이항 Tree의 재조합(Recombining)을

이 장에서 다룬 이항옵션모형에서는 옵션의 가격이 주가의 상승확률이나 하락확률과는 무관하게 결정된다. 오직 차익거래의 기회가 존재하지 않는다는 가정만이 필요하며 주가의 상승 및 하락확률에 대한 정보는 필요하지 않다. 따라서 투자자들이 주가상승배수나 주가하락배수, 그리고 무위험이자율에 대해 동질적인 기대를 갖는다면 주가의 상승 및 하락확률에 대해 서로 다른 기대를 하더라도 옵션의 균형가격은 동일하게 평가한다. 또한 옵션의 가치를 평가하기 위해 위험중립가치평가 원칙을 적용함으로써 투자자의 위험선호도와 무관하게 옵션의 가격이 결정된다. 이는 시장이 균형 상태에 있을 때에는 무위험 포트폴리오의 수익률과 무위험이자율이 같기 때문이다.

제 4 절 이항모형의 수치분석

무배당주식을 기초자산으로 하는 유럽형 콜옵션의 가치를 구하는 이항모형을 수치분석을 통해 도출해보자. 시간공간을 $i\Delta$, 상태공간(주가)을 j로 하는 2차원 격자를 구성하고, 옵션의 만기 T를 각 구간의 길이가 Δt인 N개의 구간으로 나눈다. $i\Delta$시점의 j번째 node를 (i,j)로 나타내면 $i\Delta$시점에서 주가가 취할 수 있는 상태는 $i+1$개이다. (i,j)에서의 주가는 $Su^j d^{i-j}$이고 (i,j)에서의 옵션의 가치를 $f_{i,j}$라 하자. 만기에 유럽형 콜옵션의 가치는 $\mathrm{Max}(0, S_T - E)$이므로 다음이 성립한다.

$$f_{N,j} = \mathrm{Max}(0, Su^j d^{N-j} - E), \qquad \text{for } \; j = 0, 1, \ldots, N$$

i시점의 (i,j)에서 $i+1$시점의 $(i+1, j+1)$로 움직일 확률은 p이고 $(i+1, j)$로 움직일 확률은 $1-p$이다. 따라서 위험중립가치평가의 원칙에 의하여 다음이 성립한다.

$$f_{i,j} = e^{-r\Delta t}[p f_{i+1,j+1} + (1-p) f_{i+1,j}], \qquad \text{for } \; 0 \le i \le N-1, \;\; 0 \le j \le i$$

예제

무위험 이자율 $r = 0.05$, 변동성 $\sigma = 0.3$인 시장에서 현재 KOSPI200 지수가 100인 주식을 10기간 후 100에 매입할 수 있는 유럽형 콜옵션의 가격을 구하는 C++ 코드를 만들어보자.

```
m_St[j]=m_St[j-1]*m_u/m_d;
call=max(m_St,m_E);
for(int i=m_N;i>=1;i--)
{
for(int j=0;j<i;j++)
call[j]=exp(-m_r*m_dt)*(m_p*call[j+1]+(1-m_p)*call[j]);
}
cout<<"Call price is "<<call[0]<<endl;
}

int main()
{
binomial bino(100,100,10,0.05,1.0,0.3);
bino.Get_Callprice();
}
```

결과는 다음과 같다.

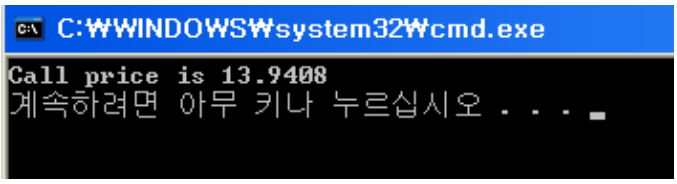


그림 9.10: 다기간 이항 모형

즉 예제의 조건을 만족하는 유럽형 콜옵션은 13.9408의 가치를 지니고 있음을 확인할 수 있다.

이항모형은 각 단계마다 기초자산의 가격이 일정한 값만큼 상승하거나 하락한다는 가정에서 출발하였다. 이러한 가정은 매우 비현실적인 것처럼 보인다. 그러나 각 단계의 시간의 크기를 작게 하여 충분히 많은 단계로 나눈 경우에는 기초자산의 가격 변동이 현실적인 분포를 갖게 된다. 실무적으로 대부분 30 ~ 50단계의 이항모형을 적용하는 것이 보통이나 이 정도가 되면 이상적인 이항분포가 거의 연속적인 분포모형으로 수렴하게 된다.

여기에서 중요한 것은 이항모형이 블랙-숄즈모형의 대체 역할 뿐 아니

제 10 장

몬테칼로 시뮬레이션 (Monte Carlo Simulation)

이 장에서는 통계적 표본추출법(Statistical Sampling)을 이용하는 몬테 칼로 기법에 대해서 알아볼 것이다. 몬테 칼로를 이용하여 주가지수 프로세스를 생성하기 위해서는 주가지수를 따르는 확률분포를 구해야 한다. 주가지수가 따르는 연속확률분포를 이산형의 난수로 대체하고 시뮬레이션을 통해 난수를 추출하여 그 난수들이 갖는 분포를 찾는다. 이러한 난수들을 원래의 주가지수가 따르는 확률분포의 함수에 대한 근사값을 구하는 것이 몬테칼로방법이다.

제 1 절 몬테 칼로 시뮬레이션의 과정

몬테칼로 방법을 이용하여 옵션가격을 결정하는 과정은 다음과 같이 요약할 수가 있다.

2. m의 소인수는 $a-1$의 인수이다.

3. 만약 m이 정수 4으로 나누어지면 $a-1$도 4으로 나누어진다.

위의 조건에 의해 모수가 결정되면 x_{i+1}를 구하고 이를 m으로 나누어 U_{i+1}를 얻는다.

$$U_{i+1} = \frac{x_{i+1}}{m}, \quad i = 0, 1, 2, \cdots .$$

다음은 선형합동법이 최대 주기를 가질 조건에 대한 증명이다.

증명

여기서 만약 m이 2의 거듭제곱일 경우에는 c가 홀수이고 $a \equiv 1 \pmod 4$인 두 조건을 만족하면 최대주기 m을 갖는다. 또한 m이 10의 거듭제곱 일 때 c는 2와 5로 나누어지지 않고 $a \equiv 1 \pmod{20}$ 의 두 가지 조건만 필요하다.

$a = 1$ 이고 c와 m이 서로소이면 당연히 주기는 m이 된다. 따라서 앞으로 $a \neq 1$임을 가정하자.

(10.1)을 사용하여 $i = 1, 2, \cdots, n-1$일 때 다음을 쉽게 얻을 수 있다.

$$x_n \equiv a^n x_0 + \frac{(a^n - 1)c}{a - 1} \pmod m.$$

그리고 우리는 $x_n = x_0$과 같이 가장 작은 n에 관심이 있다. 즉,

$$\frac{(a^n - 1)(x_0(a - 1) + c)}{a - 1} \equiv 0 \pmod m.$$

정리의 조건에 의해 $x_0(a-1) + c$는 m과 서로소이다. 따라서 다음과 같은 가장작은 n에 대해 주목하자.

$$\frac{a^n - 1}{a - 1} \equiv 0 \pmod m. \tag{10.2}$$

이제 승수a가 정리의 조건을 만족할때 가장 작은 n의 값은 m과 같음을 보이자. 첫째로, α 가 양의 정수 이고 p가 홀수인소수 일때 $m = p^\alpha$의

그러므로 j에 나타날 수 있는 p의 횟수는 $j-1$보다 적거나 같다. 그러나 $\beta \geq 1$이기 때문에 j에서 나타나는 p의 횟수는 $p^{(j-1)\beta}$항에 있는 p보다 적다. 그래서 인수 p^{α}는 j에 의해 나누어 떨어질 필요가 전혀 없다. 식 (10.4)의 우변의 모든 항은 p^{α}에 의해 나누어 진다. 이것은 적어도 제시된 조건 하에서 (10.2)는 $n = p^{\alpha}$일때 만족함을 의미한다.
우리는 이제 (10.2)를 만족시키는 p^{α}보다 작은 n은 없음을 보여야 한다. (10.2)를 만족시키는 n의 필요충분 조건은 그러한 값의 가장작은 수의 배수임을 보이는 것은 쉽다. $n = p^{\alpha}$는 (10.2)를 만족시킴을 알고있고 따라서 p의 거듭제곱인 n을 고려해야 한다. 실제로 $n = p^{\alpha-1}$은 (10.2)을 만족하지 않음을 보이면 우리의 목적에 충분하다. (10.2)의 좌변에 $n = p^{\alpha-1}$를 대입하고 a에 대해 (10.3)의 전개식을 따르면 다음을 쉽게 얻을 수 있다.

$$\frac{a^n-1}{a-1} = p^{\alpha-1} + \frac{p^{\alpha-1}(p^{\alpha-1}-1)}{1\cdot 2}kp^{\beta} + \frac{p^{\alpha-1}(p^{\alpha-1}-1)(p^{\alpha-1}-2)}{1\cdot 2\cdot 3}(kp^{\beta})^2 + \cdots + (kp^{\beta})^{p^{\alpha-1}-1}.$$

이 식의 우변이 p^{α}로 나누어지지 않음을 보일 것이다. 첫번째 항은 자명하게 p^{α}로 나누어지지 않으므로 각각의 다른 항이 p^{α}로 나누어짐을 보이면 충분하다. 증명은 위에서 주어진 (10.4)의 j번째 항의 작은 변화를 제외하고는 동일하다. 이항계수에서 나타나는 인수 p^{α}를 $p^{\alpha-1}$가 대신한다. 이때 분자에서 또 다른 인수 p가 필요하다. 이것은 p가 홀수라는 가정을 사용하면 (10.5)에서 $j-1$ 대신에 $j-2$보다 적거나 같음을 알 수 있고, 그것은 $p^{(j-1)\beta}$에서 찾을 수 있다.

우리는 이제 p가 홀수이고 $m = p^{\alpha}$일 때 정리의 증명은 완성했다. $m = 2^{\alpha}$ 일 때의 증명은 아주 조금 다르다. $\alpha = 1$일 경우에는 자명하다. $\alpha \geq 2$경우에는 위의 (10.3)에서와 다른데, 이제 양의정수 β는 1보다 커야한다. 이러한 β에 대한 제약은 다음과 같이 증명의 마지막 문장에서만 필요하다. "또 다른 인수는 $\beta > 1$의 가정을 사용하면 $p^{(j-1)\beta}$에서 찾을 수 있다."

이제 증명은 m이 소수의 거듭제곱의 제약 하에서 완성 되었고, m이 합성수일 경우로 일반화 시키는 것은 비교적 쉽다. 실제로 다음과 같이 간단하게 둘 수 있다.

```
{
R.push_back((m_a*R[i-1]+m_c)%m_M);
U.push_back((double)R[i]/m_M);
cout<<setw(9)<<i<<setw(12)<<R[i-1]<<setw(16)<<U[i-1]<<endl;
}
}

int main()
{
uniform_rand1 rand1(7,1,18);
rand1.Get_rand1();
return 0;
}
```

위의 코드 uniform_rand1.cpp를 실행한 결과는 다음과 같다.

위 결과에서 알 수 있듯이 최대주기인 $M = 18$을 갖는 유사난수를 생성했다. 19, 20, 그리고 21번째 난수는 각각 1, 2, 그리고 3번째 난수와 같다.

2.2 Non-uniform Distribution을 갖는 난수 생성: Box-Muller Method

확률밀도 함수 $f(x) = e^{-x}\chi_{[0,\infty)}(x)$에 대해서 확률분포함수는 다음과 같이 구할 수 있다.

$$F(x) = \int_0^x f(t)dt = \int_0^x e^{-t}dt = 1 - e^{-x}. \tag{10.6}$$

따라서 $F(x)$의 역함수는 $F^{-1}(u) = -\ln(1-u)$ 로 주어진다. 여기서 $\chi_E(x)$ 함수는 다음과 같이 정의된다.

$$\chi_E(x) = \begin{cases} 1 & \text{if } \ x \in E. \\ 0 & \text{otherwise.} \end{cases}$$

확률밀도 함수 $f(x) = \frac{1}{2\pi}\chi_{[0,2\pi)}(x)$에 대해서 확률분포함수는 다음과 같이 구할 수 있다.

$$F(x) = \int_0^x f(t)dt = \int_0^x \frac{1}{2\pi}dt = \frac{x}{2\pi}. \tag{10.7}$$

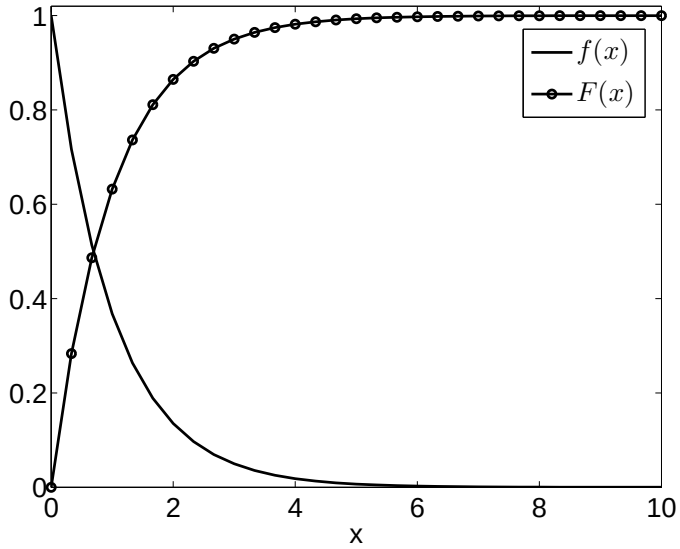


그림 10.2: 확률밀도 함수 $f(x) = e^{-x}\chi_{[0,\infty)}(x)$의 확률분포함수와 누적분포함수

2변량 표준정규분포를 따르는 점열을 구할 수 있다. 다음과 같은 극좌표들로 정규확률벡터 (x, y)를 변환시키자 (그림 10.6 참고).

$$\Omega_{xy} = \{(u, v) : -\infty < u < x, \ -\infty < v < y\}, \tag{10.8}$$

$$\Omega_{r\theta} = \{(r, \theta) : r = \sqrt{x^2 + y^2}, \ \theta = \tan^{-1}\frac{y}{x}, \ (x, y) \in \Omega_{xy}\} \tag{10.9}$$

$$x = r\cos\theta, \quad y = r\sin\theta$$

여기서 r과 θ는 다음식을 이용해서 구할 수 있다.

$$r = r(x, y) = \sqrt{x^2 + y^2}, \quad \theta = \theta(x, y) = \tan^{-1}\frac{y}{x}$$

또한, 이 극좌표변환에 의한 Jacobian은 다음과 같다.

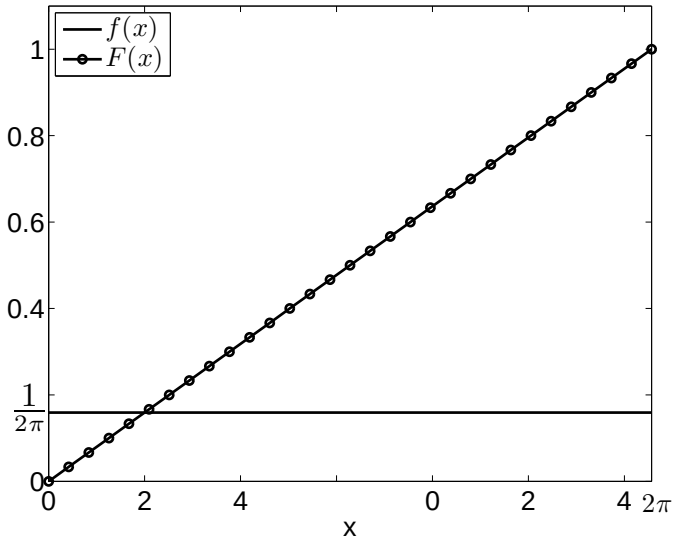


그림 10.4: 확률밀도 함수 $f(x) = \frac{1}{2\pi}\chi_{[0,2\pi)}(x)$의 확률분포함수와 누적분포함수

또한, 변수변환 $s = r^2/2$에 의해서, 위의 식을 다음과 같이 쓸 수 있다.

$$\begin{aligned} F(x,y) &= \int\int_{\Omega_{r\theta}} \frac{1}{2\pi} e^{-\frac{r^2}{2}} r dr d\theta = \int\int_{\Omega_{s\theta}} \frac{1}{2\pi} e^{-s} ds d\theta \\ &= \int\int_{\Omega_{s\theta}} f(s,\theta) ds d\theta \end{aligned}$$

여기서

$$f(s,\theta) = e^{-s}\chi_{[0,\infty)}(s)\frac{1}{2\pi}\chi_{[0,2\pi]}(\theta) \tag{10.12}$$

즉, 확률변수 s는 지수분포 $e^{-s}\chi_{[0,\infty)}(s)$을 따르고, 확률변수 θ는 균등분포 $\frac{1}{2\pi}\chi_{[0,2\pi]}(\theta)$를 따르며, 확률변수 s와 확률변수 θ는 서로 독립이다. 확률벡터 (u,v)가 정사각형 $[0,1]\times[0,1]$에서 균등분포한다고 하고, 다음 식들을 정의하자.

$$s := -\log(1-u), \quad r = \sqrt{2s} = \sqrt{-2\log(1-u)}, \quad \theta := 2\pi v$$

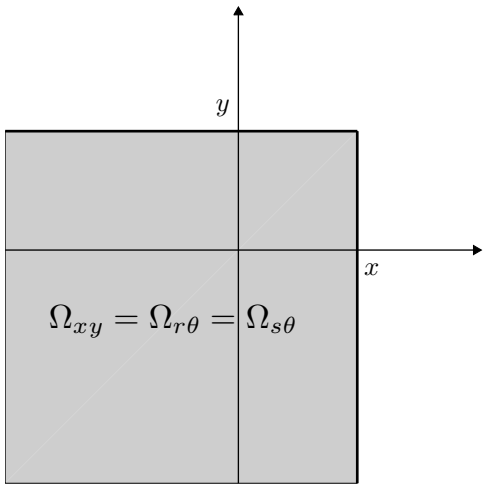


그림 10.6: 결합확률 분포함수를 계산하는 영역

```
//////////////////////+box_muller.cpp+/////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include<time.h>
#include<fstream>
#include<iomanip>
using namespace std;
#define PI 4.0*atan(1.0)

vector<double> linspace(double START,double  END, int STEP)
{
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
template<typename T>
T max(vector<T> S)
```

```
{
vector<double> u;
for(unsigned int i=0; i<A.size();i++)
u.push_back(sqrt(-2*log(A[i]))*sin(2*PI*B[i]));
return u;
}
vector<double> hist_dom(vector<double> A,int n)
{
double max_A=max(A);
double min_A=min(A);
vector<double> u=linspace(min_A,max_A,n);
return u;
}
vector<double> hist_freq(vector<double> A,vector<double> x)
{
vector<double> c;
double h=x[1]-x[0];
for(unsigned int i=0; i<x.size()-1;i++)
{
double count=0;
for(unsigned int j=0;j<A.size();j++)
{
if(x[i]<=A[j] && A[j]<x[i+1])
count+=1;
}
c.push_back(count);
}
c.push_back(1);
return c;
}

template<typename T>
bool WriteData(vector<T> data,const char* filename)
{
if (data.empty())
return false;
```

```
clc; clf; clear;
x1= load('x1_Data.dat');
x2= load('x2_Data.dat');
y1= load('A_Data.dat');
y2= load('A_Data.dat');
hold on
plot(x1, y1,'k*','linewidth',1);
plot(x2, y2,'ko','linewidth',1);
plot(x1, exp(-0.5*x1.^2)/sqrt(2*pi),'k-','linewidth',2);
legend('Z1','Z2','pdf');
xlabel('random number','fontsize',15);
ylabel('Prob.density','fontsize',15);
axis([min(x1) max(x1) 0 0.5])
set(gca,'fontsize',15)
hold off
```

2.3 복수의 기초자산을 가진 옵션의 가치 계산

앞에서 살펴본 Box-Muller 방법은 두 개 이상의 기초자산이 서로 상관관계를 갖고 있을 경우 난수를 생성하기에는 부적합하다. 이때에는 촐레스키 분해를 이용하여 기초자산간의 상관관계를 고려한 난수를 생성할 수 있다.

2.4 촐레스키 분해(Cholesky decomposition)

대칭양정치 행렬(symetric positive definite matrix)은 LU 분해의 특수한 예인 촐레스키 분해를 적용할 수 있다. 대칭양정치행렬이란 대칭 행렬 A가 **0**이 아닌 벡터 x에 대하여 $x^TAx > 0$인 행렬을 의미한다. 즉 행렬 A에 대한 고유값(eigenvalue)이 모두 양수인 행렬을 양정치행렬이라 한다. 2×2 대칭양정치 행렬 A 의 예를 들어 생각해보자.

행렬 A가 대칭양정치 행렬이라면 $A = LL^T$인 하삼각행렬 L과 상삼각행렬 L^T의 곱으로 나타낼 수 있다. 먼저, 하삼각행렬인 L을 구하기 위해

따라서,

$$L = \begin{pmatrix} \sqrt{a} & 0 \\ \frac{b}{\sqrt{a}} & \sqrt{c - \frac{b^2}{a}} \end{pmatrix}$$

이다.

2.5 기초자산간 상관관계를 반영한 난수생성

기초자산간 상관관계가 반영된 난수를 생성하기 위해서 먼저 앞서 살펴본 대로 Box-Muller 방법을 이용해서 아직 상관관계가 반영되지 않은 난수열을 생성한다. 그리고 2개 이상의 기초자산에 대한 상관계수 행렬을 촐레스키분해하여 상삼각행렬과 하삼각행렬의 곱으로 나타낸다. 그리고 앞서 구한 난수벡터에 하삼각 행렬 곱하여 상관관계가 반영된 난수를 구한다.

예로 기초자산이 2개인 경우를 생각해보자. 우선 표준정규분포를 따르는 난수 ϕ_1과 ϕ_2를 만들어보자. 상관계수가 ρ인 상관계수 행렬을 A라고 하자.

$$A = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

행렬 A를 촐레스키 분해하면 다음과 같다.

$$A = LL^T = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} 1 & \rho \\ 0 & \sqrt{1-\rho^2} \end{pmatrix}$$

분해 된 행렬 L을 이용하여 상관계수가 반영된 난수 ϕ_1^*와 ϕ_2^*로 전환하면 다음과 같다.

여기서, 독립인 확률변수들의 공분산은 0임을 알 수 있다. 상수 a에 대해서 다음 성질이 성립함을 알 수 있다.

$$\begin{aligned}
Cov[X,Y] &= Cov[Y,X] \\
Cov[X,X] &= Var[X] \\
Cov[aX,Y] &= aCov[Y,X] \\
Cov[X_1+X_2,Y] &= E[(X_1+X_2)Y] - E[X_1+X_2]E[Y] \\
&= E[X_1Y+X_2Y] - (E[X_1]+E[X_2])E[Y] \\
&= E[X_1Y]+E[X_2Y] - E[X_1]E[Y] - E[X_2]E[Y] \\
&= Cov[X_1,Y]+Cov[X_2,Y]
\end{aligned}$$

두 난수 사이에 분산을 이용하여 공분산을 구하면 다음과 같다.

$$\begin{aligned}
Cov[\phi_1^*,\phi_2^*] &= Cov[\phi_1, \phi_1\rho + \phi_2\sqrt{1-\rho^2}] \\
&= Cov[\phi_1,\phi_1\rho] + Cov[\phi_1,\phi_2\sqrt{1-\rho^2}] \\
&= \rho Cov[\phi_1,\phi_1] + \sqrt{1-\rho^2}Cov[\phi_1,\phi_2] \\
&= \rho.
\end{aligned}$$

이제 다음과 같이 정의되는 상관계수(Correlation)을 구해보자.

$$Corr[\phi_1^*,\phi_2^*] = \frac{Cov[\phi_1^*,\phi_2^*]}{\sqrt{Var[\phi_1^*]}\sqrt{Var[\phi_2^*]}} = \rho.$$

제 3 절 주가 경로(Stock Process) 시뮬레이션

몬테칼로 시뮬레이션(Monte Carlo Simulation, MC)이란 상품의 가치에 영향을 주는 변수들 간의 관계를 모형화하여 해당변수들의 미래의 값을 예측하고 이에 따라 상품의 가치를 평가하는 방법이다. 이 때 예측하고자 하는 변수들에 대해서 특정한 분포를 가정하게 되며, 해당분포를 따르는 난수를 반복적으로 발생시켜 변수의 미래 값을 예측하게 된다.

MC를 이용해서 파생상품의 가격을 결정하는 첫 단계는 기초자산의 확률과정을 모형화하는 것이다. 기초자산인 지수의 확률과정이 GBM을 따

```
vector<double> d;
double n = ((double)END - START) / (STEP-1);
for (int i=0; i < STEP; i++)
d.push_back(START+ n * i);
return d;
}
vector<double> randu(int n )
{
vector<double> u;
for (int i=0; i < n; i++) {
u.push_back((double)rand() / RAND_MAX );
}
return u;
}
vector<double> cosZ1(vector<double> A, vector<double> B)
{
vector<double> u;
for(unsigned int i=0; i<A.size();i++)
u.push_back(sqrt(-2*log(A[i]))*cos(2*PI*B[i]));
return u;
}
vector<double> sinZ2(vector<double> A, vector<double> B)
{
vector<double> u;
for(unsigned int i=0; i<A.size();i++)
u.push_back(sqrt(-2*log(A[i]))*sin(2*PI*B[i]));
return u;
}
class stock_process
{
private:
vector<double> m_S;
int m_N;
double m_r, m_vol, m_T, m_dt;
public:
stock_process(double nr, double nvol, double nT, int nN):
```

```
int main()
{
stock_process stock(0.03, 0.3, 1.0, 100);
stock.Get_Stock_Process();
return 0;
}
```

다음은 data 파일을 읽는 m-파일이다.

```
clc; clf; clear;
t = load('stock_process_x_Data.dat');
S = load('stock_process_u_Data.dat');
plot(t,S,'*-'); xlabel('Time'); ylabel('Stock Price');
```

그림 10.8 은 위의 `stock_process.cpp` C++ 코드를 실행한 결과를 나타낸 것이다.

그림 10.9 은 지수 프로세스 100개를 실행한 결과를 나타낸 것이다.

제 4 절 옵션의 payoff 계산

초기 주가지수 S_0가 주어질 때 앞의 과정을 초기시점과 만기시점 사이의 구간 $[0,T]$에 반복적으로 수행하면 만기시의 주가지수 S_T를 시뮬레이션 할 수 있다. 예를 들어 한번 시행할 때마다 유럽형 콜옵션의 만기 payoff $f_T = Max(S_T - E, 0)$을 구하고 주가지수과정을 생성하는 과정을 반복할 때마다 시뮬레이션 횟수만큼 Payoff 값들을 얻을 수 있다.

제 5 절 payoff 들의 기대값 추정

앞 절에서 구한 payoff들을 위험중립확률 Q를 이용하여 만기시의 기대값을 추정한다. 시뮬레이션을 통해 얻어진 payoff들의 총합을 시뮬레이션 횟수

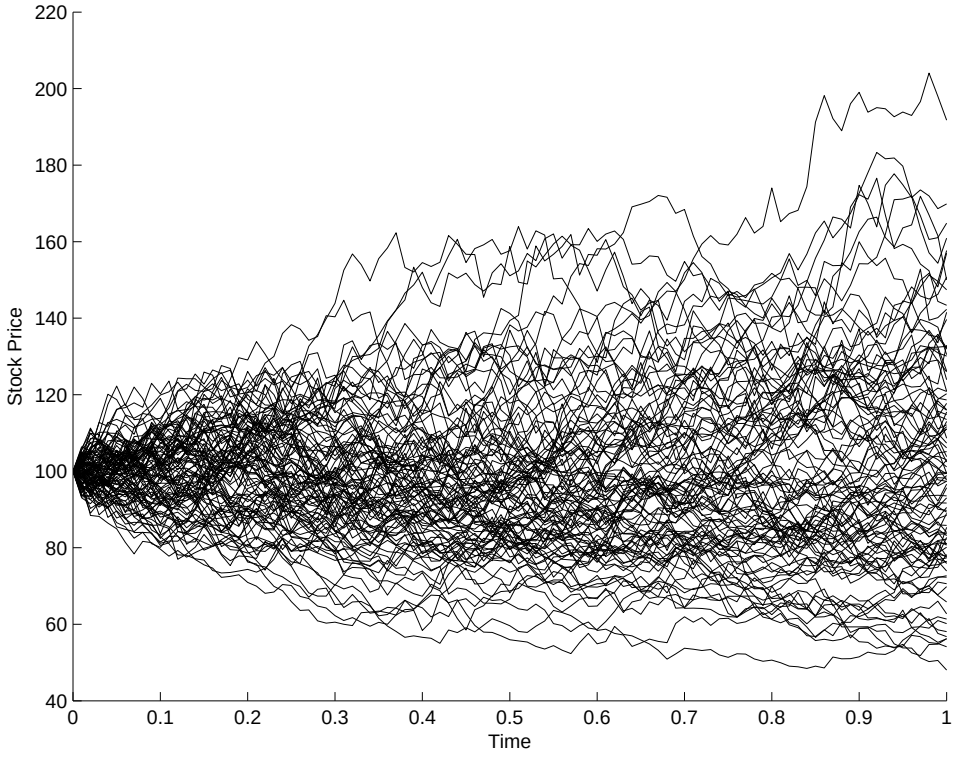


그림 10.9: 100개의 지수 과정 생성

$$f_0 = e^{-rT}[E_Q(f_T)] \quad = \quad e^{-rT}\left[\frac{1}{N}\sum_{i=1}^{N} Max(S_{T,i} - E, 0)\right]$$

제 7 절 수치 분석

파생상품 가격 결정은 먼저 MC에 의해 생성된 경로들(paths)에 따라 해당 파생상품의 payoffs를 계산하고 이에 대한 기대값을 구하여 무위험 이자율로 할인하게 되면 파생상품 가격이 도출된다. `MC_call.cpp`는 유럽형 콜옵션을 MC를 이용하여 도출하는 C++ 코드이다.

```
return P;
}
vector<double> randn2(int n)
{
vector<double> u(randu(n)),v(randu(n)),P(sinZ2(u,v));
return P;
}
class MC_call
{
private:
int m_M,m_E,m_S,m_n;
double m_r, m_vol, m_T, m_dt;
public:
MC_call(int nE,int nS,int nn,double nr, double nvol, double nT, int nM):
  m_E(nE),m_S(nS),m_r(nr), m_n(nn),m_vol(nvol), m_T(nT), m_M(nM){}
  ~MC_call(){};
  void Get_Call_price(void);
};
void MC_call::Get_Call_price()
{
srand((unsigned int)time(NULL));
m_dt=m_T/m_n;
vector<vector<double>> m_SP(m_M, vector<double>(m_n+1));
vector<double> R(m_n+1);
for(int i=0;i<m_M;i++)
m_SP[i][0]=m_S;
for(int i=0;i<m_M;i++)
{
int half=floor(m_n+1/2.0);
for(int j=0; j<half;j++)
R[j]=randn1(half)[j];
for(int j=0; j<m_n+1-half;j++)
R[j+half]=randn2(m_n+1-half)[j];
for(int j=1;j<m_n+1;j++){
m_SP[i][j]=m_SP[i][j-1]*exp((m_r-pow(m_vol,2)/2)*m_dt
+m_vol*R[j]*sqrt(m_dt));}
```

까? 정규분포($N(0,1)$)를 따르는 난수 ϕ_i $(1 \le i \le n)$가 있다고 가정했을 때,

$$\sum_{i=1}^{n} \sqrt{\Delta t}\phi_i = \phi_1\sqrt{\Delta t} + \cdots + \phi_n\sqrt{\Delta t} = \phi\sqrt{n\Delta t} \tag{10.14}$$

이 성립하기 때문이다. 즉, 전체 시간을 n개로 나누어 여러 번에 걸쳐 계산한 다음 합산한 것이나 전체 시간을 한 번에 계산하는 것이 확률적으로 동일한 값을 갖는다는 것이다.

$$\begin{aligned} E[\sum_{i=1}^{n} \sqrt{\Delta t}\phi_i] &= \sum_{i=1}^{n} \sqrt{\Delta t}E[\phi_i] = 0 \\ Var[\sum_{i=1}^{n} \sqrt{\Delta t}\phi_i] &= \sum_{i=1}^{n} \Delta t Var[\phi_i] = \sum_{i=1}^{n} \Delta t = n\Delta t \end{aligned}$$

따라서, 다음이 성립함을 확인할 수 있다.

$$\sum_{i=1}^{n} \sqrt{\Delta t}\phi_i = \sqrt{n\Delta t}\phi,$$

여기서 $\phi \sim N(0,1)$이다. 그러므로 몬테칼로 시뮬레이션을 시행할 때 대부분의 경우에 있어서 $n = 1$로 두는 것이다.

다음은 두개의 기초자산이 있는 경우에의 MC 방법에 대해서 알아보자. `MC_call2d.cpp` 은 C++ 코드이다. 이를 실행하면 다음과 같은 결과를 얻는다.

```
//////////////////////+MC_call2d.cpp+//////////////////////
#include<iostream>
#include<cmath>
#include<vector>
#include <algorithm>
#include<time.h>
#include <numeric>
#include<cstdlib>
using namespace std;
#define PI 4.0*atan(1.0)
```

```
class MC_stock_process
{
private:
int m_S,m_E,m_M;
double m_vol;
vector<vector<double>> m_SP;
vector<double> info_vec;
public:
MC_stock_process(int nS,int nE,int nM,double nvol):
  m_S(nS),m_E(nE),m_vol(nvol),m_M(nM) {}
  ~MC_stock_process(){};
  vector<vector<double>> Get_stock_process(void);
  vector<double> info(void);
};
vector<vector<double>> MC_stock_process::Get_stock_process()
{
vector<vector<double>> m_SP(m_M, vector<double>(2));
for(int i=0;i<m_M;i++)
m_SP[i][0]=m_S;
return m_SP;
}
vector<double> MC_stock_process::info(void)
{
info_vec.push_back(m_E);
info_vec.push_back(m_vol);
return info_vec;
}

class MC_call_Price2d
{
private:
vector<vector<double>> m_SP1, m_SP2,m_L;
vector<double> m_info1,m_info2,m_w0,m_w;
double m_rho,m_r,m_T,m_dt;
int m_ns;
public:
```

```
m_SP2[i][1]=m_SP2[i][0]*exp((m_r-pow(m_info2[1],2)/2)*m_dt
+m_info2[1]*m_w[1]*sqrt(m_dt));
}
vector<double> max_m_SP1,max_m_SP2,max_m_SP;
for(int i=0;i<m_ns;i++)
max_m_SP1.push_back(max(m_SP1[i][1]-m_info1[0],0.0));
for(int i=0;i<m_ns;i++)
max_m_SP2.push_back(max(m_SP2[i][1]-m_info2[0],0.0));
for(int i=0;i<m_ns;i++)
max_m_SP.push_back(max(max_m_SP1[i],max_m_SP2[i]));
double payoff=mean(max_m_SP);
cout<<"Call Price is "<<exp(-m_r*m_T)*payoff<<endl;
}

int main()
{
MC_stock_process stock1(100,100,1000,0.3);
MC_stock_process stock2(100,100,1000,0.3);
vector<vector<double>> process1(stock1.Get_stock_process())
,process2(stock2.Get_stock_process());
vector<double> info1(stock1.info()),info2(stock2.info());
MC_call_Price2d Price(process1,info1,process2,info2,0.5,0.05,0.5,1000);
Price.Get_call_price();
return 0;
}
```

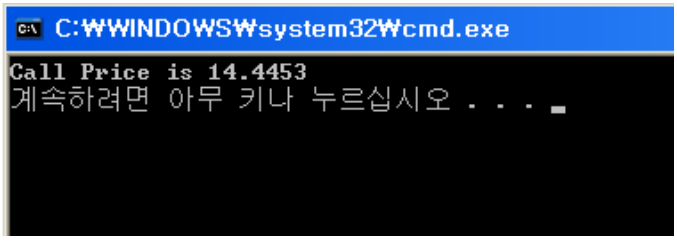


그림 10.11: 몬테 칼로를 통한 자산이 2개인 옵션 가격

참고 문헌

[1] F. Black, M. Scholes, *The Pricing of Options and Corporate Liabilities*, Journal of political economy, 1973

[2] Brandimarte P., *Numerical Methods in Finance and Economics*, Wiley, 2/E, 2006

[3] Clewlow L., Strickland C., *Implementing Derivatives Models*, John Wiley & Sons, 1998

[4] Glasserman P.,*Monte Carlo Methods in Financial Engineering*, Springer, 2003

[5] Higham D.J., *An Introduction to Financial Option Valuation*, Cambridge Univ Press, 2004

[6] Hull J.C., *Options, Futures, and Other Derivatives*, Prentice Hall, 7/E, 2008

[7] Seydel R.U., *Tools for Computational Finance*, 3/E, Springer , 2006

[8] Taleb N.,*Dynamic Hedging ; Managing Vinilla and Exotic options*,,John Wiley & Sons, 1996

[9] Wilmott P., *Paul Wilmott on Quantitative Finance*, 2/E, Wiley , 2006

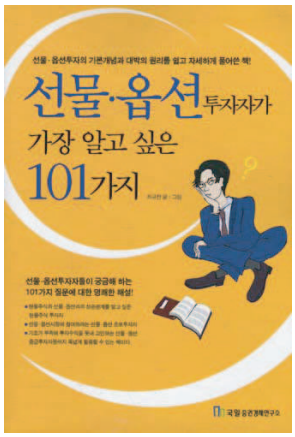
[10] Wilmott P., Howison S., Dewynne J., *The Mathmathics of Financial Derivatives*, Cambridge Univ Press, 1995

금융공학을 위한 수학 (이인형)
금융자산을 기초로 한 복합 상품인 파생상품의 이론가격결정을 위해 필요한 기본적인 수학 지식을 다룬 전공서. 확률의 기본 개념, 확률과정, 금융공학을 위한 기초수학, 가격결정 패러다임 등 6개 장으로 설명했다.

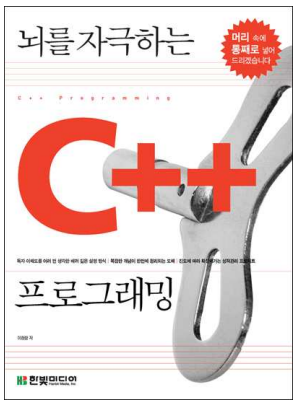
금융공학의 수학적 기초 (성승제)
금융수학 입문서. 이 책은 미적분학에서 필요한 집합론과 실해석학, 확률론과 확률해석학에 대한 내용을 담고 있다.

금융수학 (김정훈)
수리금융의 기본개념과 확률미적분학의 기초지식을 익히고 이를 금융문제에 응용하는 내용을 주요 골자로 지은 책이다.

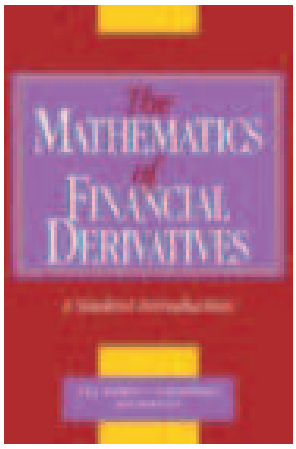
선물옵션 (이필상)
선물, 옵션, 스왑과 같은 파생 금융상품에 대한 전반 적이고 체계적인 내용을 다룬 전공서. 선물의 개념, 선물시장의 구조와 운영, 금리/통화/주가지수 선물, 스왑, 옵션시장의 구조와 운영 등 14개 장으로 설명하고 각 장마다 연습문제를 실었다.



선물,옵션 투자자가 가장 알고 싶은 101가지 (최규찬)
투자자들이 알고 싶어하는 선물 옵션거래의 핵심사항들을 간단명료하게 해설하고 있다. 투자자들이 궁금해 하는 점들을 구체적으로 예시하고 그에 대한 답변을 제시함으로써 선물 옵션거래의 원리와 기법을 쉽고 정확하게 이해할 수 있도록 했다.

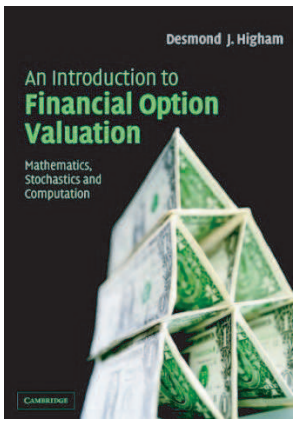


뇌를 자극하는 C++ 프로그래밍 (이현창)
명확하고 빠른 '개념'의 이해를 최우선의 목표로 삼았다. 복잡한 개념이 한번에 정리되는 그림설명이 개념의 빠른 이해를 도와주며, 수시로 예제와 실행결과를 바로 확인할 수 있어 개념을 신속하게 적용하고 실험해볼 수 있다. 이 책에서는 3단계 프로젝트가 제공되며, 파트별로 배운 문법을 총동원하여 실전에서 필요한 기능을 직접 만들어 본다.



The Mathematics of Financial Derivatives (Paul Wilmott)

Finance is one of the fastest growing areas in the modern banking and corporate world. this, together with the sophistication of modern financial products, provides a rapidly growing impetus for new mathematical models and modern mathematical methods; the area is an expanding source for novel and relevant 'real-world' mathematics.

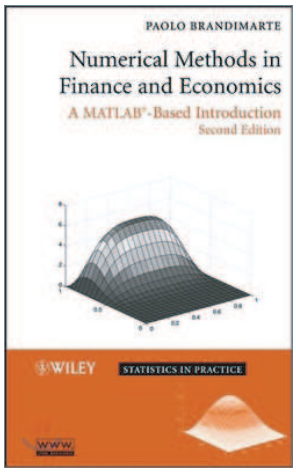


An Introduction to Financial Option Valuation (Desmond Higham)

This is a lively textbook providing a solid introduction to financial option valuations for undergraduate students armed with a working knowledge of first-year calculus, written in a series of short chapters, its self-contained treatment gives equal weight.

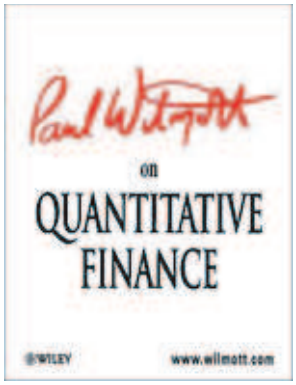
Computational Finance (Levy, George)

Computational finance presents a modern computational approach to mathematical finance within the window environment, and contains financial algorithms, mathematical proofs and computer code in C/C++. The author illustrates how numeric components can be developed which financial routines to be easily called by the complete range of Windows applications, such as Excel, Borland Delphi, Visual Basic and Visual C++.



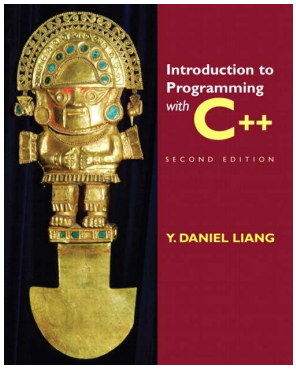
Numerical Methods in Finance And Economics (Paolo Brandimarte)

A state-of-the-art introduction to the powerful mathematical and statistical tools used in the field of finance. A MATLAB-Based Introduction bridges the gap between financial theory and computational practice while showing readers how to utilize MATLAB the powerful numerical computing environment– for financial applications.



Paul Wilmott on Quantitative Finance 3 Volume (Paul Wilmott)

Volume 3: Advanced Topics; Numerical Methods and Programs. In this volume the reader enters territory rarely seen in textbooks, the cutting-edge research. Numerical methods are also introduced so that the models can now all be accurately and quickly solved.



Introduction to Programming with C++
(Y.Daniel Liang)

There are many C++ texts. What distinguishes this book from others are the fundamentals-first approach and the writing style. The fundamentals-first approach introduces fundamental programming concepts on control statements, loops, functions, and arrays before introducing object=oriented programming.