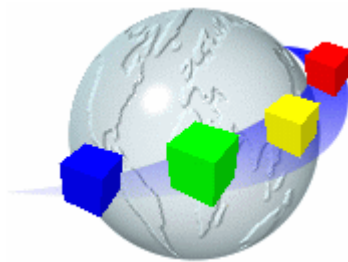


WebCab Options and Futures v2.5

WebCab
Components



Preface

This documentation accompanies the WebCab Options and Futures J2SE Component. The purpose of this documentation is to provide a clear and concise description of all aspects that are likely to be encountered in real life applications by developers and users of this enterprise Application. This class allows the use of quantitative and risk management techniques for European, Asian, American, Lookback, Bermuda and Binary Options; we also cover futures and forward contracts. We apply the Black-Scholes-Merton Options pricing model using Analytic, Monte Carlo and Finite Difference techniques. We also offer estimates of implied and historical volatility, futures account management and market risk monitoring techniques.

The first chapter of this documentation contains a brief introduction to the most important implemented features and related system requirements. In chapter two we let the developer quickly get started with deploying the component by detailing deployment techniques on the most widely used application servers. We also detail how the functionality of this component can be tested through connecting to online web service demos hosted at WebCabComponents.com. The third, fourth and fifth chapters contains the financial and mathematical documentation, which represents the theoretical background of this Applications implemented features. Chapters six and seven contains the programmer's guide, giving a road map for developers to take advantage of every feature and capability. In chapter eight we provide some examples to illustrate how features detailed within the programmer's guide can be applied in practice. Finally, in chapter ten we introduce WebCab Components, its philosophy and approach to serving the JavaTM development community with robust and powerful J2SE Components.

Good luck with your project and thank you for your interest in our components.

The WebCab Components Team

Contents

Preface	i
1 Introduction	1
1.1 Product Description	1
1.1.1 Overview	1
1.1.2 Details	1
1.2 Prerequisites and Compatibility	4
1.2.1 System Requirements	4
1.2.2 Compatibility	4
2 Where do I Start?	5
2.1 Using the J2SE JAR file	5
2.1.1 Compilation	5
2.1.2 Packaging	6
2.2 Testing the Component	9
2.2.1 Accessing an Online Demo	9
2.2.2 Compatible Web Containers	9
2.2.3 Selecting a method to test	9
2.2.4 Inputing data	9
2.3 Installing the Web Application Example Locally	10
3 Options Documentation	12
3.1 European and Binary Options	12
3.1.1 Payoff functions	13
3.1.2 Put-Call Parity	13
3.2 Trading Strategies	13
3.2.1 Spreads	14
3.2.2 Combinations	14
3.3 Volatility and how to estimate it	14
3.3.1 Standard Historical Estimate	15
3.3.2 ARCH, EWMA and GARCH models	16
3.3.3 GARCH(1,1) Model	17
3.4 Black-Scholes Model for European Options	17
3.4.1 Options on Indexes, Currencies and Futures	18

3.5	Greeks for European Options	18
3.5.1	Risk Management	19
3.5.2	Delta	19
3.5.3	Theta	19
3.5.4	Gamma	20
3.5.5	Vega	20
3.5.6	Rho	20
3.6	Implied volatility	20
4	Futures Documentation	22
4.1	Futures Contracts	22
4.1.1	Forward Contract Pay Off	22
4.2	Using Futures contracts to hedge	22
4.2.1	Hedging a portfolio using index futures	22
4.2.2	Optimal Hedge Ratio	23
4.3	Interest Bearing Investments	23
4.4	Pricing of Futures Contracts	23
4.4.1	Assumptions underlying the pricing model	23
4.4.2	Futures and Forwards on Stocks, Bonds and Indexes	24
4.5	Futures on Commodities	25
5	Exotic Options Documentation	27
5.1	Introduction	27
5.2	Types of Option Contract	27
5.2.1	Contracts with different Payoff Functions	27
5.2.2	Weak Path Dependence and American Options	28
5.2.3	Strong Path Dependence - Asian and Lookback Options	29
5.2.4	Time dependence - Bermudan options	29
5.3	The Black-Scholes Model	29
5.4	General Methods of Pricing Options in the Black-Scholes World	30
5.4.1	The Black-Scholes Equation	30
5.4.2	Boundary and initial / final conditions	31
5.4.3	Methods of solving Black-Scholes PDE	31
5.5	Finite Differencing	32
5.5.1	Single-Asset Options	32
5.5.2	Multi-Asset Options	34
5.5.3	Modifications for American Options	35
5.5.4	Modifications for Strongly Path Dependent Options	36
5.6	Monte Carlo	37
5.6.1	Random walks	37
5.6.2	Generating normal variables	39
5.6.3	Options on many underlying assets: Higher dimensions	39
5.6.4	Advantages of Monte Carlo simulations	40
5.7	Sources of Error	40

5.7.1	Errors determined by algorithm instability	41
5.7.2	Monte Carlo errors due to number of simulations used	42
5.7.3	Finite Differencing errors due to wrong boundary conditions	42
5.7.4	Finite Differencing errors due to the size of the price step	43
5.7.5	Errors due to the size of the time step	43
5.8	Greeks of Exotic Options	44
5.8.1	Finding Greeks of Exotic Options using Monte-Carlo	45
5.9	Scenario Analysis	45
5.9.1	Scenario Grids	47
5.9.2	Scenario Testing	49
6	Programmer's Guide	52
6.1	Developing with J2SE	52
6.1.1	Stand-alone Java Applications	52
6.1.2	Java Applets	54
6.2	J2EE™ Solutions Using J2SE Components	55
6.2.1	Writing JSP Pages	55
6.2.2	Designing EJB™ Components with J2SE Components	56
6.3	The Options Module Methods	58
6.4	The Futures Module Methods	59
6.5	The Exotic Options Module	59
6.6	Specifying Contract Details	59
6.6.1	Payoff Function	59
6.6.2	The Payoff Function Interface	60
6.6.3	Boundary conditions	60
6.7	Support for Developers	62
6.7.1	Compilation and Custom Modification of Source Code	62
6.7.2	Online	62
7	Examples	63
7.1	Question and Answer (QA) Client Examples	63
7.1.1	Overview	63
7.1.2	Structure of QA Examples Directory	63
7.1.3	Top Four levels of the QA Directory Structure	64
7.1.4	Bottom Two levels of the QA Directory Structure	64
7.1.5	Quick Start Guide	64
7.1.6	Explanation of the QA Directory Structure and its files	65
7.1.7	Remarks on Java compilers	66
7.2	Options Custom Examples	67
7.2.1	Volatility	67
7.2.2	ImpliedVolatility	70
7.2.3	EuropeanDelta	71
7.2.4	EuropeanTheta	71
7.2.5	EuropeanGamma	72

7.2.6	EuropeanVega	73
7.2.7	EuropeanRho	74
7.2.8	OptionStrategies	74
7.2.9	Put-Call Parity	77
7.2.10	Binary Options	77
7.2.11	Database Example with JDBC Mediator	78
7.3	Futures Custom Examples Examples	79
7.3.1	Interest	79
7.3.2	FuturesEvaluation	80
7.3.3	Forwards	81
7.3.4	FuturesHedging	82
7.3.5	FuturesOnCommodities	83
7.4	Exotic Options Custom Examples	83
7.4.1	About the examples	83
7.4.2	Testing procedure	84
7.4.3	Example 1 - European vanilla options	84
7.4.4	Example 2 - Supplying your own payoff function and boundaries	85
7.4.5	Example 3 - Supplying your own payoff function for multi-asset boundaries	86
7.4.6	Example 4 - Thorough comparison between finite differencing and Monte Carlo	87
7.4.7	Example 5 - American options	87
7.4.8	Example 6 - American multi-asset options	88
7.4.9	Example 7 - Example of Bermudan options	89
7.4.10	Example 8 - Example of Binary options	89
7.4.11	Example 9 - Example of Asian options	90
7.4.12	Example 10 - Example of Lookback options	91
7.4.13	Example 11 - Example of Monte Carlo with error control	91
8	Guide to WebCab Components	92
8.1	The Company	92
8.2	Presentation of Products	92
8.3	Supported Clients, IDEs, Containers and DBMSs	92
8.4	Transparent Functionality	93
8.5	Code Conventions	93
8.6	Company Culture and Activity	93
8.7	Product Life cycle	93
8.8	Support, Warranty and Upgrades	93

Chapter 1

Introduction

1.1 Product Description

1.1.1 Overview

Offers quantitative and risk management techniques for a wide range of option and futures contracts. We apply the Black-Scholes-Merton Options pricing model to European, Asian, American, Lookback, Bermuda and Binary Options using Analytic, Monte Carlo and Finite Difference techniques. Models of implied and historical volatility along with futures account management and market risk monitoring are also included.

1.1.2 Details

The WebCab **Exotic Options** module implements the following methods and procedures:

- **Types of Options** - Within this module we show explicitly how-to and offer practical advice on the valuation of Asian, American (single and multi-asset), Lookback, Bermuda, European (single and multi asset) and binary options using the Monte Carlo and Finite Difference techniques.
- **Finite Difference Methods** - powerful method for finding solutions of the Black-Scholes Equations.
 - **Single Asset Options** - We provide an explicit and fully implicit algorithms including a framework in which to measure stability issues under differing scenarios.
 - **Crank-Nicholson** - is a fast and stable method for evaluating single asset option contracts.
 - **Multi-Asset** - Implement a general multidimensional finite-difference algorithm.
 - **American, Bermuda Options Modification** - we apply the ‘Successive Over-relaxation’ technique in order to value American and Bermuda options.

- **Asian and Lookback** - examples of how strongly path dependent options can be evaluated using Finite Difference methods is given.
- **Monte Carlo** - can be effectively applied to value a large range of option contracts.
 - **Flow implementation** - including generation of normal variables and the simulation of the random walk and corresponding cash flows ensures that our implementation of this technique can be applied to value almost any option contract.
 - **Options on many underlying assets** - Generate correlation random variable using Cholesky factorization in order to value options contract of European type which depend on many underlying assets.
 - **Control Structure** - the user has full control over the number of simulations and/or the required precision.

The WebCab **Options** module offers the following functionality:

- **European and Binary Options** - The (Analytic) Black-Scholes model is fully implemented for European and Binary Options on stocks, currencies and indexes.
- **‘The Greeks’** - We offer methods for the evaluation of ‘the Greeks’ (delta, gamma, rho, theta, vega) for European options on stocks, indexes and currencies according to the Black-Scholes model.
- **Volatility Estimates** - the volatility may be estimated directly from historical values or from one of the following models:
 - **ARCH** - Autoregressive Conditional Heteroscedasticity model.
 - **EWMA** - Exponentially Weighted Moving Average model.
 - **GARCH(1,1)** - Generalized Autoregressive Heteroscedasticity model.
- **Implied Volatility** - Calculates the implied volatility for dividend and non-dividend paying stocks from the Black-Scholes formulae.
- **Payoff Functions** - Pay off functions at expiry for European and Binary Options are implemented.
- **Put - Call Parity relations**
 - **Put - call parity** relations for European options on an asset with no yield or a continuous yield.
 - **Put - call parity** relations for Binary options on an asset with no yield.
 - **Implied risk-free interest** - the implied risk free interest rate is calculated when either the prices of put/call European or put/pull Binary option is known.
- **Trading Strategies** - the following pay-off functions for the following option trading strategies are implemented.

- **Spread Option Strategies** - Bull Spreads, Bear Spreads and Butterfly Spreads.
- **Combination Option Strategies** - Straddles and Strangles.

WebCab **Futures** module implements the following methods and procedures:

- **Pricing on investment and consumption assets** - Pricing of futures contracts on stocks, bonds, indexes, currencies and commodities.
 - **Futures on stocks, bonds, indexes** - evaluation for assets with or without income, effective gearing.
 - **Futures on commodities** - cost of carry, utility yield.
- **Hedging** - Portfolio hedging using index futures, optimal hedge ratio.
 - **Portfolio Hedging** - delta hedge a portfolio using the beta coefficient.
 - **Optimal Hedge Ratio** - the optimal ratio of the size of the position taken in futures contracts and the size of the exposure.
- **Future Account management** - margin, daily P&L, total equity, excess margin.
- **Interest calculations** - return, compound interest, compounding periods conversion.

The Risk Management functionality included within this Component:

- **Delta Limit Monitoring** - For a portfolio (which may include Futures, Options, etc) the delta limit can be assigned and checked.
- **Scenario Analysis** - Allows for an asset or portfolio to be stressed and for the resulting behavior to be analyzed. We offer methods which stress the asset in any one or two of the underlying market variables.

This product also contains the following features:

- **GUI Bundle** - we bundle a suite of graphical user interface JavaBean components allowing the developer to plug-in a wide range of GUI functionality (including charts/graphs) into their client applications.
- **JDBC Mediator** - A J2SE Component which mediates between a J2SE component, its J2SE Clients and the Database server. The JDBC Mediator J2SE classes are a convenient way of enhancing all financial and mathematical specific methods with JDBC-based functionality. Check the `jdbc` subpackage of every J2SE class for JavaDocs documentation.
- **Web Application Example** - A Java WAR file which contains a JSP example that makes use of the functionality provided by our J2SE Component.

- **Synthetic JDBC** - The JDBC functionality provided by the Web Application example included within this package. This Web Application is an example of how to make a JSP client using our J2SE Component while manually implementing the JDBC code. The JSP Application applies J2SE methods to certain rows from the database and lists the output in HTML format.

1.2 Prerequisites and Compatibility

1.2.1 System Requirements

- An Operating System running Java™
- Pentium® III 500 MHz
- 128MB RAM

1.2.2 Compatibility

Operating System for Deployment

- Windows 2003, XP, 2000, NT, 9x
- Sun Solaris
- Linux/Unix
- IBM AIX
- HP-UX

Component Type

- JavaBeans™/Java API Library

Built Using

- Java™ 2 SDK Standard Edition 1.3.1/1.4.x

Compatible IDE Tools

- Borland JBuilder
- BEA Studio
- Sun One IDE (formally Forte for Java)
- Eclipse
- Oracle JDeveloper

Chapter 2

Where do I Start?

Start using the Options and Futures v2.5 J2SE Component right away by following the few quick and simple steps described in this chapter. If you require additional information regarding the use of this J2SE Component, then please don't hesitate to contact us via our support forum at: <http://www.webcabcomponents.com/support/index.php>

2.1 Using the J2SE JAR file

The Options and Futures v2.5 component comes wrapped inside a Java JAR file located inside the */Deployment* directory of this package¹. This class library contains all classes that provide the functionality offered by this J2SE Component as documented inside the API reference directory in this package (*/JavaDocs*). Regardless of the Java solution you are developing you will need to include this file within your project, or distribute it with your completed Application. Please note that whenever using the `J2SE-JARFile.jar` name throughout the next pages we will be referring to this Java JAR file located inside the */Deployment* directory.

2.1.1 Compilation

When compiling a Java solution that makes use of this J2SE Component you will need to make sure that the path to the J2SE JAR file is specified inside the `CLASSPATH` environment variable. The two most popular ways to compile an application are through calling the compiler via the command line or by using an IDE via JBuilder.

Compiling at the Command Line

If you compile and run your Java classes at the command prompt you should make sure that the `CLASSPATH` environment contains the path to the J2SE JAR file. In order to accomplish this under Windows you will need to type the following line at the command

¹If you are using the Demo version of this product, the name of the J2SE JAR file is *OptionsDemoJ2SE.jar*. The name of the J2SE JAR file for one or more developers and Site-wide licensed products is *OptionsJ2SE.jar*.

prompt:

```
set CLASSPATH=%CLASSPATH%; J2SEPath
```

where *J2SEPath* is the full path to the J2SE JAR file such as:

```
C:\My Downloads\Options\Deployment\OptionsJ2SE.jar
```

Under Linux the corresponding line is:

```
export CLASSPATH=$CLASSPATH: J2SEPath
```

Once you have correctly set the `CLASSPATH` you may invoke the `javac/java` tools to compile and run your Java classes. Another way to set the `CLASSPATH` variable is by directly passing its contents to these tools when typing them at the command prompt. For example, under Windows you would type:

```
javac -classpath %CLASSPATH%; J2SEPath ClassName.java
```

which would compile the *ClassName.java* file with the specified classpath, without affecting the global classpath after the compilation is done. The equivalent for the Linux `javac` tool is:

```
javac -classpath $CLASSPATH: J2SEPath ClassName.java
```

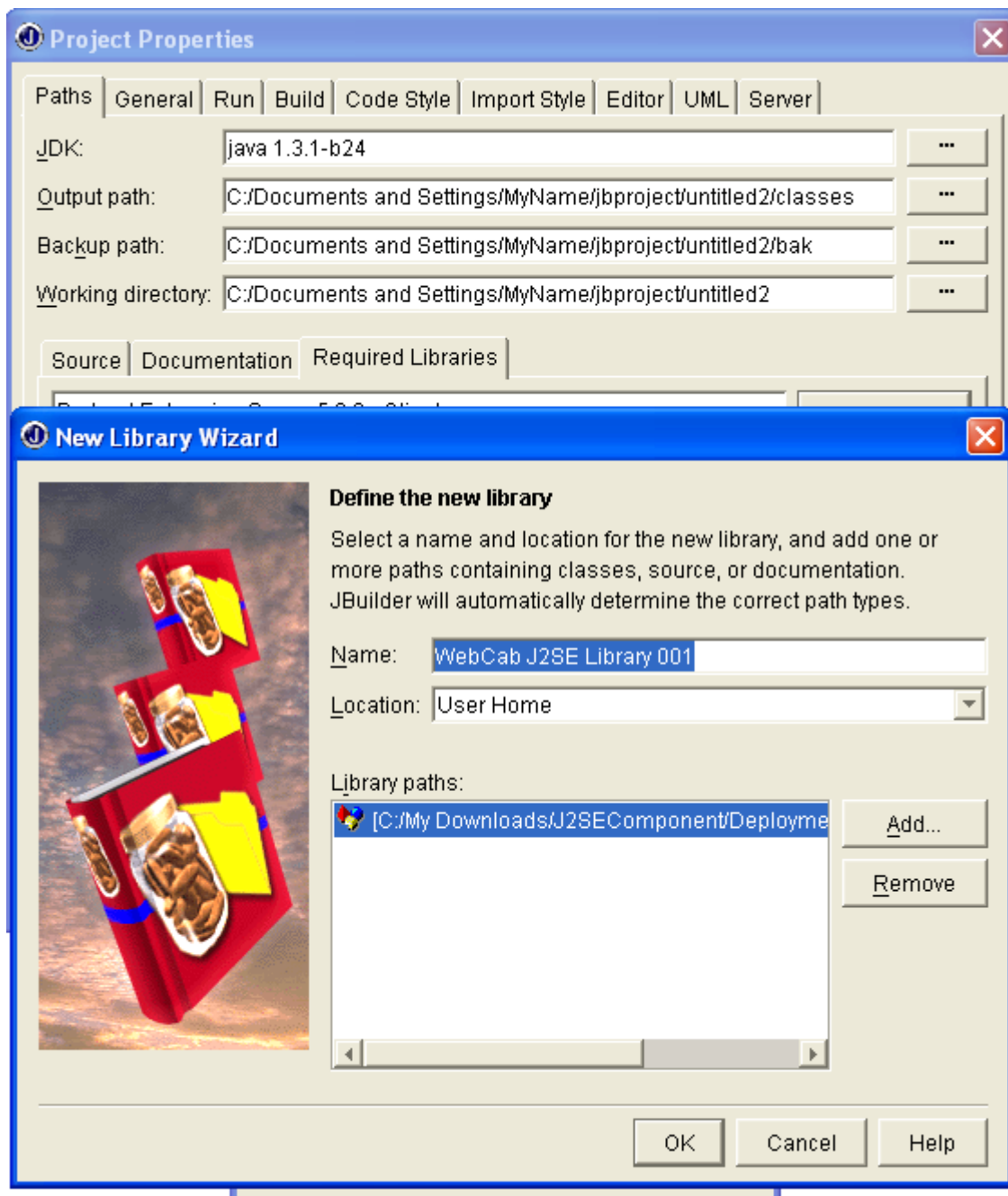
Compiling in Borland's JBuilder™

In order to use the functionality provided by our J2SE Components within your JBuilder™ projects, you will need to add the J2SE JAR file located inside the */Deployment* directory of this package to the project's list of required libraries.

Open the **Project Properties** dialog from the **Project** menu and select the **Required Libraries** tab. Click the **Add...** button, and then click the **New...** button in the newly opened dialog. Type in a distinctive name for this library and click the **Add...** button. Pick the J2SE JAR file from its location by browsing through the directory structure and press **OK**. As soon as you close each dialog by pressing the **OK** button you will be ready to compile and run your Java project together with the J2SE Components.

2.1.2 Packaging

Certain types of Java solutions require special packaging such as JAR, WAR or EAR files. For example, by packaging our JAR file within an WAR archive the JAR file may be deployed onto a Web container such as Tomcat. In order to deploy this component onto a J2EE Application server you may provide the JAR within either a WAR or EAR archive.



Adding a J2SE JAR file to the list of libraries required by a Borland JBuilder project.

Java Applets

It is always recommended that a Java Applet be archived together with its referenced class files in a Java JAR file which can then be downloaded and unpacked on the client's machine. If your Java Applet uses functionality provided by any of the classes of our J2SE Component, you will need to unpack the J2SE JAR file and repackage it with all the other files the Applet requires to run properly. In order to unpack the J2SE JAR file you will

type at command prompt:

```
jar xf J2SE-JARFile.jar
```

Then you can take the unpacked files and folders and add them to your Java Applet JAR file inputting the command:

```
jar cfM Applet-JARFile.jar <classfiles-and-resources>
```

JSP/Servlet Pages

Deployment of JSP Pages onto an Application Server is done through a special archive called Web Archive (WAR). If your JSP pages make calls to our J2SE Component, you should add the J2SE JAR file to the *WEB-INF/lib* subdirectory of the deployment WAR file. For this purpose you should create a directory named WEB-INF and a subdirectory named WEB-INF/lib, where you should put a copy of the J2SE JAR file. Then run the following command, adjusting the name of the WAR file accordingly:

```
jar uf JSP-WARFile.war WEB-INF
```

You may then deploy the Web Application as usual.

J2EE Deployment

When deploying a complete Enterprise Application which contains enterprise modules that require the presence of the classes provided by this J2SE Component, you should make sure to include the J2SE JAR file to the Enterprise Application Archive (EAR) and add a Class-path entry to the manifest file of the modules that reference this component.

First, create a file named `manifest.tmp` which contains the following:

```
Manifest-Version: 1.0
Class-Path: J2SE-JARFile.jar
an empty line
```

Then, run the following command:

```
jar ufm MODULEFile manifest.tmp
```

where `MODULEFile` is the name of the module which uses the functionality within the J2SE JAR file (for example `SessionBean.jar`). The final step is to package the EAR file as usual and run the following command:

```
jar uf J2EE-EARFile.ear J2SE-JARFile.jar
```

After performing all these steps the enterprise archive should be ready for Application Server deployment.

2.2 Testing the Component

2.2.1 Accessing an Online Demo

By far the easiest way to test the functionality of this J2SE Component is to view the online demo. The online demo can be accessed from our [J2SE Homepage](#) by clicking the '[Demo]' link corresponding to this Application.

2.2.2 Compatible Web Containers

This demo runs inside an online web container and implements the Servlet and Java Server Pages (JSP) technologies. The demo can be accessed through a web browser and is compatible with Internet Explorer 5, Netscape Navigator 4, Netscape 6, Opera 5 and higher.

2.2.3 Selecting a method to test

After clicking on the '[Demo]' link for this application from our home page the J2SE Web demo will launch within your web browser. To order to select a method from this application's J2SE component use the drop-down menu on the left hand side of the screen to navigate through all implemented functions. The menu lists all the J2SE components on the first level and their corresponding functions on the second and third level. Click on the plus icon in order to expand the J2SE component menu and left click a function to select it.

2.2.4 Inputing data

The selected function will be displayed in the center of the screen accompanied by its description and parameter characterization. Once the nature of the method is clear you may test the method by inputing the parameters within the text boxes provided or associating a database table field using our database management tool.

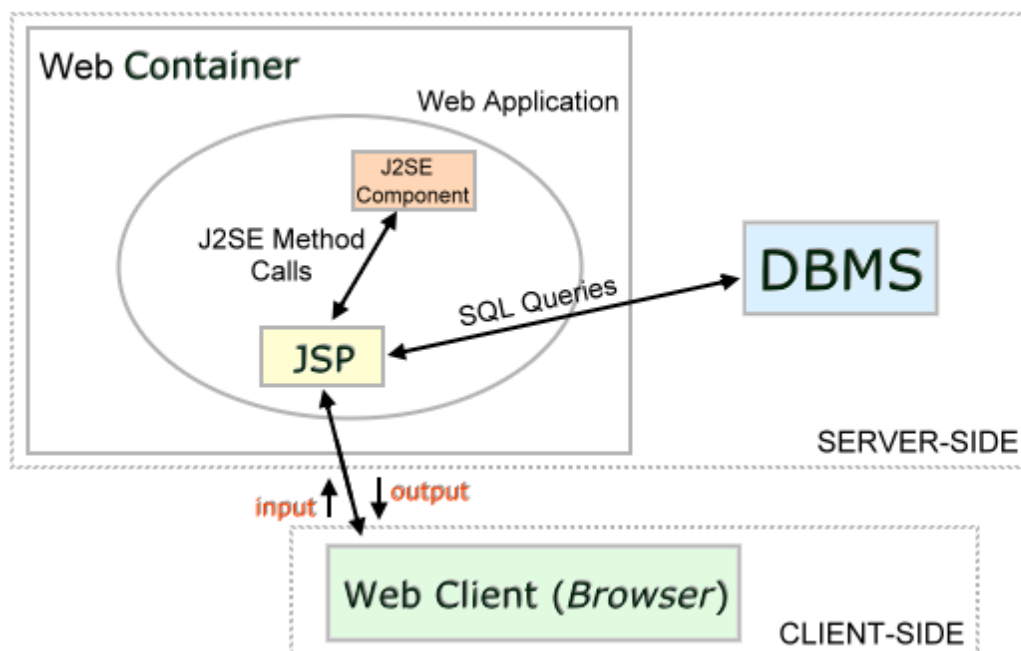
Running the demo using text boxes

In case you have chosen to input parameters by value type each number in the corresponding box while paying attention to each parameter description. When all the parameters has been input, press the "Get Result" button on the right-hand side and towards the bottom of the screen in order to request the solution from the deployed J2SE component.

If by chance any parameters are out of range you will be prompted to enter a correct value before proceeding.

Running the demo using Synthetic JDBC

If you choose to run in Synthetic JDBC mode, you will be asked to enter your DBMS connection properties, such as username and password. Next screen you will be able to choose a column for each parameter by browsing it directly from your the database. Press the the “Get Result” button to generate a report on the input columns and their corresponding computed result. The Synthetic JDBC concept is illustrated below:



www.webcabcomponents.com

Simplified exemplification of how the Synthetic JDBC works within a Web Browser.

2.3 Installing the Web Application Example Locally

This package provides an additional Web Application example which makes use of the functionality provided by this J2SE Component. This example comes wrapped inside a Java WAR file located in the */Deployment/Jsp Examples* directory of this package.

In order to test this Web Application you will first need to follow Web Container specific deployment steps, usually consisting in copying the Java WAR file to one of the Web Server's subdirectories. After deployment has completed, open a Web browser and type one of the following addresses that correspond to your Web or Application Server.

- IBM WebSphere - <http://localhost:9080/OptionsWebExample>
- BEA WebLogic - <http://localhost:7001/OptionsWebExample>

- Oracle 9iAS - <http://localhost:8888/OptionsWebExample>
- Sun ONE Application Server - <http://localhost:81/OptionsWebExample>
- Borland AppServer - <http://localhost:8080/OptionsWebExample>
- Ironflare Orion - <http://localhost:8888/OptionsWebExample>
- JBoss - <http://localhost:8080/OptionsWebExample>

Remark This J2SE Web Application Example has also been deployed on our server. You can access it online by [clicking here](#).

Chapter 3

Options Documentation

An option is the right to trade a particular asset at some time or interval in the future at an agreed price. The mechanism for determining the agreed price and the variety with which the option can be exercised leads to the large range of options which exist within financial markets.

Within Options theory there are a number of terms which are repeated used. Below we define the use of these terms once and for all so that they can be used within the rest of this documentation and within the JavaDocs without confusion.

- Premium - the amount paid for the option contract
- Underlying (asset) - the financial instrument which the options depends upon, denoted by S
- Strike (or exercise) price - the amount with which the underlying can be brought (call) or sold (put) during the period when the option can be exercised, denoted by E_c, E_p , respectively
- Expiration date - the date of expiry of the option contract, denoted by T
- Intrinsic value - the payoff that would be received if the underlying is at its current level when the option expires
- Time value - the value in excess of the intrinsic value

3.1 European and Binary Options

A European (or Vanilla) option is one of the following two types:

- Call option: a call option is the right to buy an asset S_c , at an agreed price E_c , at a fixed time T_c
- Put option: a put option is the right to sell an asset S_p , at an agreed price E_p , at a fixed time T_p

We will also consider binary options (or digitals) which are contracts which have a payoff at expiry which is discontinuous in the underlying asset price. We consider a **binary call option** which has a given fixed payoff if the underlying asset is greater or equal to the strike price at expiry. The binary put option is similarly defined to have a fixed payout if the underlying asset is less than the strike price at expiry.

3.1.1 Payoff functions

The payoff function of an option contract is the cash (or equivalent) settlement which the investor receives at expiry. The payoff functions for an European option at expiry are given by:

$$\text{Call Option payoff function} = \max(S_c - E_c, 0)$$

$$\text{Put Option payoff function} = \max(E_p - S_p, 0)$$

respectively.

We also provide the payoff functions for put and call Binary options.

3.1.2 Put-Call Parity

By arbitrage arguments one is able to derive what is generally referred to as put-call parity relations. This allows for example the European call option price to be derived from the European put option price when the two options act on the same underlying asset with the same strike and expiry date. Similarly the European put option price can also be derived from the European call option. The case when the underlying assets pay no dividend and pays a continuous dividend are treated for European options. In a similar fashion we also offer parity relations for binary options on assets which do not pay a dividend during the life of the option.

In order to derive the call (or put) options value for a put (respectively call) option with the use of parity relations we need to know the (fixed or average) risk free interest rate during the life of the option contract. Within our component we allow the user to derive the implied risk-free interest rate when the put and call option values are known in the following instances:

- European options on an asset with zero yield
- European options on an asset with continuous yield
- Binary options on an asset with zero yield

3.2 Trading Strategies

An option trading strategy generally involves the use of more than one option which displays a preferred payoff function characteristic from the investors stand point. Here we

discuss a number of trading strategies which have become widely used within various financial markets.

3.2.1 Spreads

A spread trading strategy involves taking a position in two or more options on the same underlying asset and with the same expiry date. Procedures for the evaluation of the payoff functions at maturity of the following spreads are given:

- Bull Spreads - Buying a call option with strike x_1 , and selling a call option on the same asset but with a strike x_2 , where $x_1 \leq x_2$. The individual entering into this contract will gain if the underlying asset rises in value.
- Bear Spread - Buying a call option with strike x_1 , and selling a call option on the same asset but with a strike x_2 , where $x_1 \geq x_2$. The individual entering into this contract will gain if the underlying asset falls in value.
- Butterfly Spread - This trading strategy involves the use of three options with different strikes. To construct the spread the investor must buy a call option with a relatively high strike x_1 , and a relatively low strike price x_2 , and sell two options with a strike midway i.e. at $(x_1 + x_2)/2$. Generally, the butterfly spread is constructed with the midpoint close to the present market value. In this case the investor will gain if the price at expiry is close to its present price and will take a small loss if the price moves significantly in either direction.

3.2.2 Combinations

A combination is an option trading strategy which involves taking positions in both put and call options on the same underlying asset. Methods for the evaluation of the payoff of the following strategies are given:

- Straddle Combination - involves taking a long position in a call option and a long position in a put option with the same strike price and expiration date.
- Strangle Combination - consists of a long position in a call option and a put option with different strike prices and the same expiry date.

3.3 Volatility and how to estimate it

The volatility is a measure of the degree to which a given quantity fluctuates, which is in a sense the amount of randomness of that parameter. The volatility is most commonly first encountered in the context of the volatility of an asset's price in order to be applied to the pricing of European options in the context of the Black-Scholes model (see below). Even in the context of European options and the Black-Scholes model that is the (constant) volatility of the underlying asset price there are a number of senses in which the volatility will be referred to, namely:

- **Historical Volatility** - Measure of the past fluctuations of the underlying asset price. Below we describe the most common historical measure of the volatility used which is the standard deviation of the log of the price returns.
- **Implied Volatility** - The volatility implied from the price of the European option in accordance with the Black-Scholes model. However, in general the implied volatility could be deduced from any option price in accordance with any (volatility based) option pricing model.
- **Forecast Volatility** - An estimate of the future volatility of the underlying asset price over some period. This is a required parameter within option pricing models.

It is important to point out that there is no one true measure of volatility. Moreover, the measure should be tailored to take into account its application. For example, within the Black-Scholes model we which to know as a model parameter the average volatility over a given period, whereas when evaluating value-at-risk (VaR) or performed portfolio optimization (CAPM) we are interested in the present value of the volatility of the assets within the portfolio.

3.3.1 Standard Historical Estimate

The historical volatility (as mentioned above) is most commonly estimated as the standard deviation of the log of the asset returns. That is, if

- S_i = the asset price at the end of the i th day
- D_i = the interest payment (or dividend) from the asset on the i th day
- n = the number of days over which the historical data is taken

Then the log of the return on the i th day is given by:

$$u_i = \ln \left(\frac{S_i + D_i}{S_i} \right)$$

for $i = 1, 2, \dots, n$, and the **historical daily volatility estimate**, using the past n days observations of the asset price is given by:

$$\text{Daily Volatility} = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (u_i - \bar{u})^2} \quad (3.3.1)$$

where $\bar{u}$ is the arithmetic average of the returns u_i 's, over the n days.

Remarks If we had considered the returns on the underlying asset over the past n months then the corresponding formula for (3.3.1) would express the historical volatility per **month**. Similar remarks hold if other time periods are used.

We are able to express the historical volatility derived from the n asset prices with respect to another time period. In particular, we are able to derive the annual volatility.

The standard error per annum of the (annual) volatility historical volatility estimate is given by:

$$\frac{s^*}{\sqrt{2n}}$$

where s^* is the annual volatility.

Our Implementation

This estimation procedure has been implemented within the class as the following two methods:

- `Volatility.historicalEstimate` - Estimates the historical volatility of an asset which does not pay dividends or interest payments.
- `Volatility.historicalEstimateWithDividends` - Estimates the historical volatility of an asset which does pay dividends or interest payments.

The volatility evaluated by the historical procedure can be rescaled by the procedures:

- `Volatility.yearDaysRescaling` - Rescaling annual volatility to volatility over a given number of days.
- `Volatility.daysYearRescaling` - Rescaling the volatility over a number of days to the annual volatility.

We also provide the following error estimate:

- `Volatility.historicalEstimateStandardError` - The standard error of the historical annual volatility estimate.

3.3.2 ARCH, EWMA and GARCH models

These models were introduced to produce more accurate estimates of the present volatility from historical data. With these more refined models more weight is given to recent measurements and an aspect of reversion to mean is incorporated which is displayed by volatility in real world situations is incorporated within the models.

Within the volatility class we implement all of these models offering more refined means by which to estimate and predict the volatility. The full names for these models are:

- ARCH - autoregressive conditional Heteroscedasticity
- EWMA - exponentially weighted moving average
- GARCH - generalized autoregression conditional heteroscedasticity

ARCH Model

The ARCH model was first suggested by R.Engle, see ‘Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of UK Inflation,’ *Econometrica*, 50 (1982), 987-1008. This estimate is based on the long term volatility (v) and historical observations (s_i).

EWMA Model

This model is a special case of the ARCH model. The weights assigned to the historical values decrease exponentially. J.P.Morgan uses the EWMA model for updating daily volatility estimates in its RiskMetrics database. In particular, if the weights are $\{\alpha_i : i \in 1, \dots, n\}$, then the relation between the weights is given by $\alpha_{i+1} = \lambda \alpha_i$, where λ is a constant between zero and one. J.P.Morgan found that across a range of market variable, by taking $\lambda = 0.94$, the forecasts of the volatility rate came closest to the realized volatility.

3.3.3 GARCH(1,1) Model

The GARCH(1,1) model was introduced by Bollerslev in 1986, see ‘Generalized Autoregressive Conditional Heteroscedasticity’, *Journal of Econometrics*, 31(1986),307-27. The EWMA model is a special case of the GARCH(1,1) model. The GARCH model exhibits mean reversion whereas the EWMA model does not and hence the GARCH model is theoretically more appealing than the EWMA model.

3.4 Black-Scholes Model for European Options

We consider the Black-Scholes model for the evaluation of European put and call options at time zero on assets with or without interest payments. In particular, for the European put (denoted by ‘p’) and call (denoted by ‘c’) options there exists closed formulae given by:

$$\begin{aligned} c &= S_0 N(d_1) - X e^{-rT} N(d_2) \\ p &= X e^{-rT} N(-d_2) - S_0 N(-d_1) \end{aligned}$$

where

$$\begin{aligned} d_1 &= \frac{\ln(S_0/X) + (r + \sigma^2/2)T}{\sigma\sqrt{T}} \\ d_2 &= d_1 - \sigma\sqrt{T} \end{aligned}$$

and $N(x)$ is the normal probability distribution with zero mean and standard deviation of 1. The constant S_0 is the stock price at time zero, X is the strike of the options, r is the continuously compounded risk free interest rate, σ is the volatility and T is the time to maturity of the option.

The case when there exists dividends is handled by assuming that the stock price consists of a riskless component of the known dividend payments during the option period

and a risky component which accounts for the remainder of the option price. By discounting the dividend payments by the risk free interest rate we can deduce the present value of the option taking into account the effect of dividend payments which will decrease the value of the underlying asset. Strictly speaking the volatility used for this method should be the volatility of the stock price net the present value of the assets dividend.

3.4.1 Options on Indexes, Currencies and Futures

European options on indexes, currencies and futures contracts are also traded on exchanges and within the over-the-counter market. By extending the classical Black-Scholes model for stocks we are able to evaluate options on this wider range of underlying assets.

These methods cover the evaluation of the following contracts:

- Index options on the S&P100 and S&P500 which are traded on the Chicago Board Options Exchange
- Currency Options trading on the Philadelphia exchange including Dollar-Yen call and put with have expiration dates in the nearest three months
- Futures options of Brent Crude Futures traded on the International Petroleum Exchange (IPE)

3.5 Greeks for European Options

The Greeks describe the rate of change of the option price with respect to various market variables. These include the underlying stock price (delta), time (theta), rate of change of the delta with respect to the underlying stock price (gamma), volatility (vega) and the interest rate (rho). We will consider the Greeks for European options where the underlying asset may yield a continuous dividend. We use the Black-Scholes model for the derivation of all the Greeks which is based upon following assumptions:

- An asset price follows a Markov stochastic process known as geometric Brownian motion
- The short selling of securities with full use of proceeds is permitted
- There are no transaction costs or taxes. All securities are perfectly divisible
- There are no riskless arbitrage opportunities
- Security trading is continuous
- The risk-free interest rate is constant and the same for all maturities

Remark The Black - Scholes model can be extended in a number of directions by weakening these assumptions. In particular, we will consider the case when the asset has a continuous yield.

3.5.1 Risk Management

The Greeks provide a convenient mechanism for financial institutions to monitor and control the risk they are exposed to from outstanding options contracts. Options can also be an important tool for the hedging of risk associated with stock (or other types of) portfolios. One of the most popular techniques is known as Delta hedging which involves taking an options position with a delta which is the negation of the portfolios delta. Then the collection of assets consisting of the portfolio and the options will have zero delta or a delta-neutral position. The delta function is a linear functional which mean that the delta of a portfolio is the sum and the deltas of the individual components. In particular, if an options portfolio has a delta of Δ , and consists of the w_i options α_i , for $i = 1, \dots, n$, then:

$$\Delta = \sum_{i=1}^n w_i \Delta_i$$

where Δ_i , is the delta of the option α_i .

Once a portfolio has been made delta neutral, the next stage is to consider its gamma and by taking an off setting position try to attain gamma neutrality. In a similar way a trader may wish to hedge a portfolio against volatility changes and may taken an option position which create an overall vega neutral position. In practice, options traders usually rebalance their portfolios at least once a day to maintain delta neutrality. It is usually not feasible to maintain gamma and vega neutrality on a regular basis due to the trading costs which would be incurred. Typically, a trader monitors these measures, and if they get too large, either some of this excess is hedged away or trading is curtailed.

3.5.2 Delta

Within our component we offer methods for the calculation of the delta of a European call and put option on an asset with or without a continuous yield. These methods can not only be applied to the calculation of the delta of options on stocks but also to options on indexes, currencies and futures. We offer methods for each of these situations.

3.5.3 Theta

The theta of a portfolio or asset is the rate of change of its value with respect to the decrease in the time to maturity. We calculate theta in time measured in years. Usually, when theta is quoted time is measure in days. Theta can be converted to a per day measurement from a per year measurement by dividend by 252 the number of trading days in a year.

We consider the theta of a stock which may or may not pay of continuous dividend. We also provide methods for the evaluation of the theta of a European option on currencies, indexes and futures contracts.

3.5.4 Gamma

The gamma of an portfolio of derivatives on an underlying asset is the rate of change of the portfolios delta with respect to the price of the underlying asset. In short, it is the second derivative of the portfolio value with respect to the asset price. Gamma is useful for determining the sensitivity of delta to changes in the underlying asset and hence the relative frequency in which the portfolio will need to be rebalanced in order to stay approximately delta neutral.

We provide methods for calculating the gamma of options on stocks, indexes, currencies and futures.

3.5.5 Vega

Vega is a measure of the sensitivity of the portfolio to changes in the volatility. It may seem strange to use the Black - Scholes model to calculate vega since within this model the volatility is assumed to be constant. Ideally it would be better to develop a model where the volatility is assumed to follow a stochastic process. However, it turns out that the vega calculated from a stochastic volatility model is very similar to the Black - Scholes vega so the practice of calculating vega from a model where volatility is constant works reasonably well.

3.5.6 Rho

The Rho of a portfolio of derivatives is the rate of change of the portfolio value with respect to the risk free interest rate. In the case of options on currencies where there are two interest rates there are two 'rho's', one for the base currency interest rate and the other for the foreign currencies interest rate.

3.6 Implied volatility

The one parameter within the Black-Scholes pricing formulae that cannot be directly observed is the volatility of an asset. We showed above how the volatility can be estimated from an assets historical price data. An alternative approach is to derive the volatility of an asset from the market value of options on that given asset. The estimate for the volatility derived by this method is referred to as the implied volatility.

Unfortunately, it is not possible to invert the Black-Scholes equations so that σ is expressed as a function of S_0 , X , r , T and c . However an iterative search procedure can be used to find the implied σ . In our component we use **Ridders** method (see our component WebCab Equation Solver for more details) in order to find the implied volatility with a precision to six decimal places (i.e. $10e - 6$).

Implied volatilities can be used to monitor the market's aggregate view of the volatility of a particular quoted asset. Analysts will often calculate implied volatilities from actively traded options on a given asset and use these results to calculate the price of less actively traded options on the same underlying asset.

It is important to note that the prices of deep-in-the-money and deep-out-of-the-money options are relatively insensitive to volatility. Therefore the implied volatility calculated from these options tends to be unreliable.

Chapter 4

Futures Documentation

Within this chapter we detail the type and nature of the implemented models within the Futures module. We will deal with quantitative and risk management techniques with futures and forward contracts on stocks, indexes, commodities and currencies.

4.1 Futures Contracts

A futures contract is an agreement to buy or sell an asset at a certain future time for a fixed price. Futures contracts are usually traded through an exchange with the profit or loss from the futures contract held through a broker calculated every day and the according cash flows paid from one party (or account) to the other.

Futures contracts have become standardized for different markets. We will consider within the section futures contracts on stocks, bonds, commodities, currencies and indexes.

4.1.1 Forward Contract Pay Off

A forward contract is closely related to a futures contract the main difference being that no money changes hands between the parties until the expiry date. We provide the payoff function which calculates the cash flow between the parties on the expiry date.

4.2 Using Futures contracts to hedge

All methods related to the use of futures contracts to hedge positions can be found within the bean `FuturesHedgingBean`.

4.2.1 Hedging a portfolio using index futures

Stock index futures can be used to hedge the risk in a well diversified stock portfolio. Since by the Capital Asset Pricing model we know that the relationship between the expected return on a well diversified portfolio of stocks and the return on the stock market as a

whole is described by the parameter β (beta). The beta is the slope of the line of best fit when the excess return of the portfolio is regressed against the excess return on the market over the risk free rate. That is, when $\beta = 1$, the returns from the portfolio tend to match the return from the index. If $\beta = n \in (0, +\infty)$, then the returns from the portfolio tend to be n times the return from the index.

Within this component we include methods which allow the beta of a portfolio to be estimated and the calculation of the number of index futures contracts necessary to hedge a well diversified portfolio. We also offer a method which returns the number of index futures contracts which need to be brought or sold in order for the β , of the portfolio constructed portfolio to be modified to any value we choose.

4.2.2 Optimal Hedge Ratio

The optimal hedge ratio is the ratio of the size of the position taken in futures contracts to the size of the exposure in order to have a perfectly hedged portfolio. Practically it may not be possible to perfectly hedge a physical position with futures contracts since the exposure may not be equal to an integer multiple of the contract size. A method which calculates the number of futures contracts which offers the best hedge is given.

4.3 Interest Bearing Investments

The evaluation of futures contract is closely related to the analysis of interest bearing investments. For this reason we include basic methods related to fixed interest paying deposits with compound and continuously compounded interest.

Later we will consider more delicate interest rate models which allows the interest rate curve to be non-constant.

4.4 Pricing of Futures Contracts

With the pricing of futures contracts it is important to distinguish between futures on investment assets and futures on consumption assets. This distinction is necessary because the holder of consumption asset utility may depend mainly on issues completely independent from the actually profit and loss realized from holding the contracts. For this reason for consumption assets a non-arbitrage argument cannot be used in order to derive a pricing methodology.

4.4.1 Assumptions underlying the pricing model

In order to construct a pricing model we need to assume that there exists some market participants for which the following is true:

- These market participants are not subject to transaction costs when they trade

- These market participants are subject to the same tax rate on all net trading profits
- The market participants can borrow money at the same risk-free rate of interest as they can lend money
- The market participants take advantage of arbitrage opportunities as they occur

These assumptions are not so unreasonable since there exists the set of large investment banks which broadly speaking satisfy these requirements. In particular, investment banks will often have in-house arbitrage trading desks or specialists covering all the markets in which they have significant customer flow. Hence since these investment banks will take advantage of these arbitrage opportunities almost as soon as they arise means that in practice arbitrage opportunities disappear almost as soon as they arise. An implication of these assumptions is therefore that market prices are such that there is no arbitrage opportunities.

4.4.2 Futures and Forwards on Stocks, Bonds and Indexes

We provide the following methods for the evaluation of future contracts on stocks and bonds:

- `futurePriceNoIncome` - this method returns the value of a futures contract on an underlying asset which does not pay an income
- `futurePriceWithIncome` - evaluates the futures price today on an asset which pays a known income during the life of the contract. This could be a futures contract on a bond or dividend paying stock.
- `futuresPricesWithDividend` - evaluates the future price today on an asset which has a continuous yield.
- `shortForward` (`longForward`) - evaluates the value of an short (respectively long) forward contracts residual value
- `futureOnIndex` - returns the price today of a futures contract on an index
- `futureOnCurrencies` - returns the price today of a futures contract on currencies
- `gearing` - we also include a method which calculates the effective gearing of a futures position
- `costOfCarry` - we calculate the cost of carry for an investment asset

Because an exchange traded futures contract is marked to market on a daily basis it cannot accrue a residual value.

Within an environment of constant interest rates the value of a futures contract and the value of a forward contract are the same. When interest rates are allowed to vary as happen in the real world these two contracts will generally have different values.

4.5 Futures on Commodities

Futures on commodities have a number of different features. These features are related to the fact that a holder of a futures contract of a commodity may not hold the contract for investment purposes. That is, certain individuals may wish to continue to hold a futures contract (or the underlying commodity) even if there are arbitrage opportunities. If there is not a significantly large number of market participants who will arbitrage then arbitrage arguments to evaluate these futures will produce misleading results.

With futures on commodities the settlement of the contract is generally in the underlying physical asset and hence the seller of a futures should hold the physical commodity in order to be certain that they can settle the contract at expiry. Hence storage costs of the underlying commodity, and its availability will effect the futures prices.

We consider commodities which are held primarily for investment purposes for example gold and commodities which are held mainly because of their consumption value for example oil. We assume that for commodities of investment type that a section of the holders (for example investment banks) of the futures contracts and underlying assets will always arbitrage if the opportunity arises. We also assume that these investors have the following properties:

- These market participants are not subject to transaction costs when they trade
- These market participants are subject to the same tax rate on all net trading profits
- The market participants can borrow money at the same risk-free rate of interest as they can lend money

We refer to commodities which do not satisfy these conditions as consumption commodities. Though large investment banks still operate within these markets it may not be possible to short the underlying assets and go long on a futures contract. This is because holders of the underlying commodity hold the asset for reasons of consumption and may be reluctant to sell the commodity and buy futures contracts because futures contracts cannot be consumed. Alternatively, if an investment bank holds the underlying commodity they can always sell this asset and buy futures contracts.

We will also use the two related notions of convenience yield and cost of carry. The convenience yield measures to markets implied benefit from holding the physical commodity. The convenience yield can be viewed as the yield over the risk free interest rate which the market is prepared to pay for the convenience of holding the physical commodity. The cost of carry measures the storage cost plus the interest that is paid to finance the asset less the income earned on the asset.

We provide methods which return a value of a futures contract on an investment commodity and give an inequality which the price of a future on a consumption commodity

must satisfy. We also provide methods for the calculation of the cost of carry and convenience yield for investments and consumption commodities. Further, for an consumption commodity we offer methods which relate the cost of carry and the convenience yield.

Remark With most commodity markets the notion of the spot price does not really exist because the delivery of the physical commodity always takes some time, usually a number of weeks. Hence within our methods what we are really doing is deriving relationships between two futures prices. In particular, within our methods I refer to the rather vague notion of “commodity price” rather than the spot price of the commodity so that no confusion arises and we do not give a force impression of the way in which commodity markets work.

Chapter 5

Exotic Options Documentation

5.1 Introduction

Within this chapter we discuss firstly the definition of various exotic options and then show how one can go about pricing the analysing such option contracts. We cover American, Bermuda, Asian, Loopback and multi-asset options using Finite Differencing and Monte-Carlo techniques.

5.2 Types of Option Contract

An option is a contract between two parties in which one party buys the option contract which the other party creates (also known as writing an option contract). The option itself is the right to obtain a pay-off as defined within the option contract at a single or range of future dates.

Example: The first and simplest type of options introduced is known as a European (Vanilla) Option which gives the holder the (optional) right to trade (either buy or sell) an asset at an agreed future time at an agreed price (known as the exercise price or strike price). The issuer of the European Option has the obligation to take the other side of the trade (to the holder of the option) on the asset at the agreed price, when the holder decides to exercise the option.

Options are traded on exchanges and regulated OTC markets throughout the world. To date option contracts have been written on stocks, stock indices, currencies, debt instruments, commodities, futures contracts and even options on options.

5.2.1 Contracts with different Payoff Functions

Within this subsection we give three examples of types of option contracts which differ only in there *payoff* function.

Vanilla or European Options

The *payoff* function on a Vanilla option is given by:

- Vanilla call option payoff: $\max(\text{asset_price_at_expiry} - \text{strike_price}, 0)$
- Vanilla put option payoff: $\max(\text{strike_price} - \text{asset_price_at_expiry}, 0)$

Binary or digital Options

An increasingly important type of option is the binary or digital options. These contracts have a payoff that is discontinuous in the underlying asset price.

Option strategies

Standard combinations are option contracts are traded, and are known as option strategies. Commonly used strategies include:

- *Spread* is an option strategy consisting of a long position and a short position.
- *Bull spread* is a spread where the long position has a smaller strike price than the short position. It benefits from a rising (bull) market.
- *Bear spread* is a spread where the long position has a greater strike price than the short position. It benefits from a falling (bear) market.
- *Strangle* is an option strategy consisting of one call and one put option.
- *Straddle* is a special kind of strangle, where the strike prices for the call and the put are equal.

5.2.2 Weak Path Dependence and American Options

As mentioned before an option contract can have the property that it can only be exercised on a given date or a range of dates. Option which can only be exercised on a given date are known as European whereas options which can be exercised on a range of dates are known as **American options**. American options allow the holder the option of choosing to exercise the options *early* before the final expiry date.

American type options will depend on the path of the underlying asset and this dependency is known as *weak*. This refers to the fact that the partial differential equation derived from the Black-Scholes model has the same independent variables as the corresponding European contract. That is, there are no more variables which depend on the path taken by the underlying asset.

5.2.3 Strong Path Dependence - Asian and Lookback Options

Strongly path dependent options such as Asian and Lookback options have a payoff function which explicitly uses a new path dependent variable. For example the payoff in the case of Asian options is the dependent on the average of the underlying asset over a certain period. In this cases the following new path dependent variable will be introduced:

$$I = \frac{1}{T} \int_0^T S d\tau \quad (5.2.1)$$

where T is the time to expiry and τ is the time from inception. And the payoff function will take the following form:

$$\text{payoff} = P(S, I) \quad (5.2.2)$$

where P is some function of the underlying asset price S and the introduced variable I .

Options with a payoff dependent on the average of the asset price are called **Asian options**. Options where the payoff function is dependent on the maximum or minimum reached by the underlying asset price are called **Lookback options**.

Remark A characteristic of strongly path dependent options is that the partial differential equations resulting from the Black-Scholes model can only be solved with the introduction of a new independent variable corresponding to a path dependent value.

5.2.4 Time dependence - Bermudan options

Another variation of an option contract is when the payoff of the contract can vary with time. A typical example of such an option is a Bermudan option. A Bermuda option contract permits early exercise (as in American options) but it restricts this possibility to certain dates or certain periods.

Such options are strongly path dependent and there payoff function will take the following form:

$$\text{payoff} = P(S, I, t) \quad (5.2.3)$$

where P is some function of the underlying asset price S , I is the introduced variable depending on the path of the asset (see prior subsection) and t is the time.

5.3 The Black-Scholes Model

The payoff function is the value of the option at expiry whereas we require to know the option value now, that is, the value before expiry. Clearly any option pricing model will need to make assumptions concerning the future evolution of the price. The Black-Scholes model assumes that price evolve in accordance with the following Brownian motion known as a lognormal random walk:

$$dS = \mu S dt + \sigma S dX \quad (5.3.4)$$

where dS is the ‘infinitesimal’ change in the underlying assets price, μ is the known as the drift of the asset, dt is the time variation, σ is the volatility of the underlying asset and dX is a random variable drawn from a Normal distribution with mean zero and variance dt .

Simply knowing that the price has a lognormal evolution is not sufficient in being able to value option contracts. In order to be able to achieve this we need to introduce a further assumption which allow us to assign a *fair* value to any option contract (at least in principle). This principle is known as the ‘**principle of no arbitrage**’ and said the following:

In the absence of arbitrage opportunities (which is assumed to be the case within efficient markets), the return from a risk free portfolio must be the risk free interest rate.

5.4 General Methods of Pricing Options in the Black-Scholes World

5.4.1 The Black-Scholes Equation

If we assume the validity of the two principles:

- lognormal evolution
- no arbitrage

then it follows that the price evolution must obey the following partial differential equation (PDE):

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0, \quad (5.4.5)$$

where σ is the volatility, r is the risk-free interest rate and S is the asset price.

The linear parabolic PDE (5.4.5) is known as the *Black-Scholes Equation* and is the basis from which we will price option contracts. To price an option contract involves finding the solution of this equation for some given initial conditions.

Remark The unknown within the equation is not a value, but a function: $V(S, t)$ which stands for the value of the option at a time t , with the current asset price being S .

The Black-Scholes equation knows nothing about what kind of option we are valuing, whether it is a call or a put, American or Asian, nor what strike price is. These points are dealt with by the following final condition which with boundary conditions (see below) will ensures that we have one and only one solution to the corresponding Black-Scholes equations:

We must specify the option value V as a function of the underlying at the expiry date: $final(S) = V(S, 0)$.

This function is of course the payoff function which as mentioned above is the one thing which specifies many different types of option contracts.

5.4.2 Boundary and initial / final conditions

Knowing the Black-Scholes equation and the payoff function is generally not enough information to determine the price of an option contract. In order to select the unique pricing function we must prescribe *boundary conditions* and an *initial* or final condition of the *Black-Scholes Equation*.

The boundary conditions which are required in order to select the pricing function using the Black-Scholes Equations will naturally be the functions which describe the (expected) behaviour of the pricing function. The only problem is to select boundary conditions of the pricing function which are sufficient to uniquely determine the pricing function and yet ones which we are confident will not select the ‘wrong’ function from the solution set.

Using Extreme Values to provide boundary conditions

The asset values at which the pricing function is estimated from which we are able to provide boundary conditions are usually ‘extreme values’ with respect to the option contract. That is, the pricing function is estimated when the underlying asset price is either significantly above the strike price (for a call option this is referred to as being *deep in the money*) or is significantly below the strike price (for a call option this is referred to as being *deep out of the money*).

At ‘extreme values’ the pricing function will either be approximately zero in the case of deep out of the money, or approximately the difference between the price and the strike compounded at the risk free rate until maturity in the case of deep in the money. Since in the case of deep out of the money the option is very unlikely to be exercised whereas deep in the money options are very unlikely to expiry worthless.

In particular, within financial applications it is usually possible to specify the behavior of the pricing function (or solution) at $S = 0$ and as $S \rightarrow \infty$. We must also describe the initial conditions (i.e. when $t = 0$) that is the price for which the equation evolves. If the final condition is required then the payoff function at expiry for differing values of the underlying asset can be evaluated. By specifying these boundary conditions we will insure that the Black-Scholes equation yield a unique solution which will be the pricing function.

5.4.3 Methods of solving Black-Scholes PDE

We now have a well posed PDE problem for which there exists a unique solution. The problem now is to find this solution.

The Black-Scholes PDE is a parabolic partial differential equation (also referred to as equations of *heat* or *diffusion* type). Though there exist approaches which attempt to solve the Black-Scholes equations analytically, such as¹:

- Transformation to Constant Coefficient Diffusion Equation
- Use of Green's Functions
- Series Solution
- Fourier Transform
- Laplace Transform

For most option contracts the resulting Black-Scholes equation does not exhibit an analytic solution. In these cases you are forced to use numerical techniques in order to find the pricing function.

The two most widely used and successful numerical approaches to finding solutions of the Black-Scholes equations are **Finite Differencing** and **Monte-Carlo** techniques which we describe in the following sections.

5.5 Finite Differencing

The finite difference algorithm essentially relies on a *discretization* of the continuous PDE. The difference equations resulted from this process are used to generate a *mesh* or a *grid* which contains the discrete version of the solution. You are then able to interpolate the (discrete) solution at the grid nodes in order to find a solution over the whole region.

5.5.1 Single-Asset Options

The Explicit Algorithm

When there is only one asset involved, the Black-Scholes PDE has exactly the form presented in the section above. This can be immediately translated into a finite difference equation:

$$\frac{V_j^{n+1} - V_j^n}{\Delta t} + rj\Delta S \frac{V_{j+1}^{n+1} - V_{j-1}^{n+1}}{2\Delta S} + \frac{1}{2}\sigma^2 j^2 \Delta S^2 \frac{V_{j+1}^{n+1} - 2V_j^{n+1} + V_{j-1}^{n+1}}{\Delta S^2} = rV_j^{n+1} \quad (5.5.6)$$

where Δt is the time step, ΔS is the asset price step and where V_β^α 's denote the function values at points of the grid, where the superscripts indices $\alpha = n, n + 1$, denotes the corresponding time axis value and the subscripts $\beta = j - 1, j, j + 1$, denote the grid point on the asset price axis.

¹A detailed presentation of these methods can be found in **Cranck, JC**: Mathematics of Diffusion, Oxford, 1989; or **Carslaw, HS & Jaeger, JC**: Conduction of Heat in Solids, Oxford, 1989

Remark The grid in this case is bidimensional. Where one axis represents the time and the other axis represents the underlying asset price.

We still have the final condition which we can use since we already know $V_j^{time_steps}$ for all $1 \leq j \leq asset_price_steps$. From the equation we can immediately find the value of $V_j^{time_steps-1}$ for all $2 \leq j \leq asset_price_steps-1$, the values $V_1^{time_steps-1}$ and $V_{asset_price_steps}^{time_steps-1}$ being known from the boundary conditions. In the same manner we can compute the values for $time_step - 2$, then for $time_step - 3$ and so on, until the whole grid is evaluated.

General stability issues

The problem with the explicit algorithm approach is that it tends to amplify errors, that is it tends to be unstable. Very small and unavoidable errors caused by the limited precision of the floating point format can become exponentially larger in subsequent iterations and completely compromise the results.

It can be deduced that in order to endure the stability of the explicit algorithm, the following relationship between the time step and the asset price step must hold:

$$\Delta t \leq \frac{\Delta S^2}{2a} \quad (5.5.7)$$

where a is some constant.

Moreover, this relationship is tight meaning that if this inequality is not satisfied then the algorithm is unstable. A consequence of this relationship is that if we want to improve the accuracy by halving the asset step size within the grid, then we must reduce the time step by a factor of four. Resulting in the computation time required to complete the computation being increased by a factor of eight. The improvement in accuracy we would get from such a reduction in step sizes is a factor of four since the explicit finite-difference method is accurate to $O(\Delta t, \Delta S^2)$.

Remark There is also a lower bound for the asset price step, but this usually does not make much difference in practice unless the volatility is very small.

To conclude, the explicit algorithm is unstable if the value of Δt is too large in relation with the value of ΔS , and if ΔS itself is too small.

Fully implicit

The implicit algorithm which we will cover in this section solves the problem of stability which effected the explicit algorithm above. The fully implicit algorithm, uses the following difference equation (equivalent with the first in the sense that they both can be used to

represent the Black-Scholes PDE):

$$\frac{V_j^{n+1} - V_j^n}{\Delta t} + rj\Delta S \frac{V_{j+1}^n - V_{j-1}^n}{2\Delta S} + \frac{1}{2}\sigma^2 j^2 \Delta S^2 \frac{V_{j+1}^n - 2V_j^n + V_{j-1}^n}{\Delta S^2} = rV_j^n \quad (5.5.8)$$

using the notation defined above.

Unlike the explicit case, the solution here is not so straightforward. We cannot simply solve one equation at each iteration, starting from the final condition to complete the grid. Here we must solve a full set of *asset_price_steps* equations, which can be computationally intensive. But the implicit algorithm also has the advantage that it is unconditionally stable. This allows us to apply the algorithm without any restriction on the time step. For example, the asset price step can be small and the time step large without the method running into stability problems.

Cranck-Nicholson

The main advantage of the explicit method (see above) is that it's fast, but it also has the disadvantage that it can be unstable. In the case of the implicit method (see above section), its main advantage is that it is stable, but with the drawback that it can be slow. The Crank-Nicholson method is a hybrid of these two approaches and actually manages to combine the speed of the explicit method whilst keep the stability of the implicit algorithm. In fact, the Cranck-Nicholson algorithm is stable and yet it is also faster than the fully implicit method. Key extra ingredient is in the way it takes the finite difference approximation of the partial derivatives: which can be viewed as the average between the explicit approximation and the implicit one. The finite difference equation for Cranck-Nicholson is given by:

$$\begin{aligned} & \frac{V_j^{n+1} - V_j^n}{\Delta t} + \frac{1}{2}rj\Delta S \frac{V_{j+1}^{n+1} - V_{j-1}^{n+1}}{2\Delta S} + \frac{1}{2}rj\Delta S \frac{V_{j+1}^n - V_{j-1}^n}{2\Delta S} + \\ & + \frac{1}{2}\sigma^2 j^2 \Delta S^2 \frac{V_{j+1}^{n+1} - 2V_j^{n+1} + V_{j-1}^{n+1}}{\Delta S^2} + \frac{1}{2}\sigma^2 j^2 \Delta S^2 \frac{V_{j+1}^n - 2V_j^n + V_{j-1}^n}{\Delta S^2} = rV_j^{n+1} \end{aligned} \quad (5.5.9)$$

using the notation defined above.

Which results in the Cranck-Nicholson method having an accuracy of $O(\Delta t^2, \Delta S^2)$. For further details concerning the Crank-Nicholson method we refer the interested reader to:

Cranck, J. & Nicholson, P.: A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type. Proceedings of the Cambridge Philosophical Society **43** 50-67, 1947.

5.5.2 Multi-Asset Options

If the option contract involves more than one asset, then the original single asset Black-Scholes theory is modified with the use of concept of *correlated lognormal random walks*:

$$dS_i = \mu_i S_i dt + \sigma_i S_i dX \quad (5.5.10)$$

where S_i is the price of the i -th asset, $1 \leq i \leq n_assets$, μ_i and σ_i are the drift and volatility of that asset respectively and dX_i is a random number drawn from a Normal distribution with mean 0 and variance dt . From the definition of the Normal distribution we know that:

$$E[dX_i] = 0 \text{ and } E[dX_i^2] = dt \quad (5.5.11)$$

where E is the expectation. An extra parameter (which is in fact a matrix) in the multi-asset case is the correlation matrix between the underlying assets which the option depends on.

Correlation Matrix

The correlation matrix can be constructed from the set of pairs of historical times series of the underlying assets. This can be expressed by stating that the random numbers dX_i and dX_j are correlated. Hence:

$$E[dX_i dX_j] = \rho_{ij} dt \quad (5.5.12)$$

where ρ_{ij} is the correlation coefficient between the i -th and the j -th random walks. The symmetric matrix defined by the entries ρ_{ij} , which represent the entry in the i -th row and j -th column is called the *correlation matrix*.

Black-Scholes Equation for Multi-asset options

In this case the Black-Scholes equation for multi-asset options is given by:

$$\frac{\partial V}{\partial t} + \frac{1}{2} \sum_{i=1}^{n_assets} \sum_{j=1}^{n_assets} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^{n_assets} (r - D_i) S_i \frac{\partial V}{\partial S_i} - rV = 0 \quad (5.5.13)$$

How these equations are solved?

The numerical algorithm used to solve such an equation is a generalization of the explicit method presented above. The difference is that now the grid is not bidimensional but multidimensional. So the storage space required and the computational power needed grows correspondingly. In practice the multidimensional finite-difference algorithm cannot be used for more than three assets. In cases with more than three assets you should use instead Monte-Carlo methods which are much faster when the number of assets (i.e. dimensions) is higher than three.

5.5.3 Modifications for American Options

Finding the price of an American option, that is an option which allows early expiry, via the Black-Scholes equations is an example of what is known as a *free boundary problem*. With free boundary problems we are required to solve partial differential equations where the value of the solution at the boundary is not fixed by a boundary condition.

With American options price evaluation we can assume that both the options value and its delta ($\frac{\partial V}{\partial S}$) are continuous with respect to the payoff function, this is the called *smooth pasting principle*². Since the smooth pasting condition is satisfied whatever pricing mechanisms of American Option we use, it must be satisfied by the finite-differencing approach. Therefore, when applying finite differencing techniques to American options we modify the formula that generates the next line of the pricing grid in the explicit algorithm so that it guarantees the solution will be continuous with respect to the payoff and moreover that the continuous delta constraint will also be satisfied and so on...

Note: American options should only be priced using finite-difference methods, since Monte-Carlo techniques are inappropriate for this type of option.

5.5.4 Modifications for Strongly Path Dependent Options

Since an asset's path cannot be continuously measured, resulting option contracts which are strongly path dependent only require measurement of path dependent quantities at discrete time intervals. In particular, there is a minimum time step between sampling of path dependent quantities such as the price.

After each time step, the path dependent value must be updated to reflect the changes in the state of the asset's price path. The path dependent quantity is updated at each *sampling date* t_i and takes the value I_i for $t_i \leq t \leq t_{i+1}$, where:

$$I_i = F(S(t_i), I_{i-1}, i) \quad (5.5.14)$$

This updating rule can be generalized as the new value of I , is determined by only the old value of I , the value of the underlying at the sampling date and the date itself. If we consider for example Asian options using an arithmetic average, the updating formula will become:

$$I_i = \frac{1}{i} S(t_i) + \frac{i-1}{i} I_{i-1} \quad (5.5.15)$$

For max-type lookback options the updating formula will be:

$$I_i = \max(S(t_i), I_{i-1}) \quad (5.5.16)$$

As we said in the definition of strongly path dependent options, the option value $V(S, I, t)$; is now a function of three variables:

$$V(S, I, t) \quad (5.5.17)$$

where S is the value in the underlying asset, I is the value of path dependent variable and t is the time.

²This principle states that the value of an American Option is maximized by an exercise strategy that makes the option pricing function and its derivative with respect to the price (i.e. the option delta) continuous.

Remarks

1. The stochastic differential equation for I is degenerate (i.e. $dI = 0$) at $t \neq t_i$ for any i , because it is only updated on the discrete set of dates t_i . Therefore, provided we are not on a sampling date, the quantity I is constant and the pricing equation is simply the basic Black-Scholes equation. Though the pricing function $V(S, I, t)$, is still a function of three variables, the I variable can be effectively treated as a parameter.
2. Since the options value across a sampling date is continuous we impose at each t_i , the following condition:

$$V(S, I_{i-1}, t_i^-) = V(S, I_i, t_i^+) \quad (5.5.18)$$

where the left hand side of the expression represents the limit of $V(S, I_{i-1}, t)$, as t approaches t_i , from below; and the right hand side of the expression represents the limit of $V(S, I_i, t)$, as t approaches t_i , from above. This condition is known as the *jump condition*, and roughly says that V cannot have discontinuities in the 't' direction at the grid points.

5.6 Monte Carlo

The Monte Carlo technique is clearly very powerful and general. It can be effectively applied to a large range of option contracts. In particular, the basic method readily carries over to exotic and path dependent option contracts (e.g. Asian, American and Loopback Options). The Monte Carlo method applied to the pricing of an option contract will need to make the following steps:

- Simulate the random walk and the corresponding cash flows
- Estimate the average payoff and calculate its present value

5.6.1 Random walks

The fair value of an option in the Black-Scholes world is the present value of the expected payoff at expiry under a risk-neutral random walk of the underlying asset.

$$dS = rSdt + \sigma SdX \quad (5.6.19)$$

where r is the drift rate, S is the stock price, t is a time variable, σ is the volatility and X is the random variable with respect to the standard normal distribution.

We can therefore write:

$$\text{OptionValue} = \exp(-r(T - t))E[\text{payoff}(S)] \quad (5.6.20)$$

where r is the drift rate (as before), T is the time to expiry of the option, t is the present time, E is the expectation with respect to the risk-neutral random walk of a given payoff

(i.e. payoff(S)) being achieved at the expiry of the option contract.

Hence, we can estimate the value of an option contract by the following steps:

1. Simulate the risk-neutral random walk as discussed below, starting at today's value of the asset S_0 , over the required time horizon. This time period starts today and continues until the expiry of the option. This gives one realization of the underlying price path.
2. For this realization calculate the option payoff.
3. Perform many more such realizations over the time horizon.
4. Calculate the average payoff over all realizations.
5. Take the present value of this average which gives the option value.

The initial part of the algorithm requires first of all the generation of random numbers from a standardized Normal distribution. This will be discussed in the next section. Then one has to update the asset price at each time step using these random increments.

For the lognormal random walk we have the following simple and exact time-stepping algorithm. We write down the risk-neutral stochastic differential equation for S in the form:

$$d(\log S) = \left(r - \frac{1}{2}\sigma^2\right) dt + \sigma dX \quad (5.6.21)$$

When by integrating this equation we find:

$$S(t) = S(0) \exp \left(\left(r - \frac{1}{2}\sigma^2\right) t + \sigma \int_0^t dX \right) \quad (5.6.22)$$

So over the time step δt , we have:

$$S(t + \delta t) = S(t) + \delta S = S(0) \exp \left(\left(r - \frac{1}{2}\sigma^2\right) \delta t + \sigma \sqrt{\delta t} \phi \right) \quad (5.6.23)$$

The critical issue in this approach is that we can approximate cases where the options payoff function depends on the historical price of the underlying asset and possibly other parameters. This is because in such cases for a small δt , the above 'equality' approximately holds and so we can split the time interval into small periods and generate estimates of the options contract price.

In the case when the option only depends on the data at expiry then the payoff function will not depend on the time and other variables and the above expression will hold for all δt . In such cases we can simulate the final asset price in one giant leap to expiry, using the time step T . European options are such an instance and if we apply the Monte Carlo method then we will obtain estimates which will converge to the corresponding value given by the Black-Scholes formulae as the number of approximations is increased.

5.6.2 Generating normal variables

Most programming languages (including Java and C#) incorporate routines for sampling random numbers between zero and one. An approximate sample from a univariate standardized normal distribution $\phi(0, 1)$, can be obtained using the formula:

$$\epsilon = \sum_{i=1}^{12} R_i - 6 \quad (5.6.24)$$

where the R_i are independent random numbers between zero and one where ϵ is the required sample from $\phi(0, 1)$. This approximation is satisfactory for most purposes since it has a mean of zero, a standard deviation of one, and a third moment of zero.

5.6.3 Options on many underlying assets: Higher dimensions

Monte Carlo is a natural method for the pricing of European-style contracts that depend on many underlying assets. Supposing that we have an option with a payoff function which depends on the prices of several underlying assets $S_1, S_2, \dots, S_d$. Instead of using the Black-Scholes methods which require us to solve a PDE in $d + 1$ variables we could use the simulation methods provided by Monte Carlo procedures. In order to use Monte Carlo all we need to do is simulate:

$$S_i(t + \delta t) = S_i(0) \exp \left(\left(r - \frac{1}{2} \sigma_i^2 \right) \delta t + \sigma_i \sqrt{\delta t} \phi_i \right) \quad (5.6.25)$$

The catch is that the ϕ_i are correlated, that is:

$$E[\phi_i \phi_j] = \rho_{ij} \quad (5.6.26)$$

Generating correlated random variables

Let us suppose that we can generate d *uncorrelated* Normally distributed variables $\epsilon_1, \epsilon_2, \dots, \epsilon_d$. We can use these variables to get correlated variables with the transformation

$$\phi = M\epsilon \quad (5.6.27)$$

where ϕ and ϵ are the column vectors with ϕ_i and ϵ_i in the i -th row. The matrix M is special and must satisfy:

$$MM^T = \Sigma \quad (5.6.28)$$

with Σ being the correlation matrix.

The decomposition of the correlation matrix into the product of two matrices is not unique. One solution is the **Cholesky factorization** which produces a matrix M that is lower triangular.

5.6.4 Advantages of Monte Carlo simulations

The Monte Carlo technique is very powerful and general. All you need to do is to simulate the random walk and the corresponding cash flows, estimate the average payoff and take its present value.

However this method has also disadvantages. One of them is that it is slower compared with the finite-differencing solution of a partial differential equation. Generally speaking this is true for problems up to three or four dimensions. *When there are four or more stochastic or path dependent variables the Monte Carlo method becomes relatively more efficient.* Another disadvantage is that it is time consuming to apply this method to American options.

Because Monte Carlo simulation is based on the generation of a finite number of realizations using series of random numbers, the value of an option derived in this way will vary each time the simulations are run. If the standard deviation in the option using a single simulation is ϵ then the standard deviation of the error after N simulations is $\epsilon/\sqrt{N}$. In other words to improve the accuracy by a factor of 10 we must perform 100 times as many simulations.

5.7 Sources of Error

All numerical algorithms including the ones used here (i.e. Monte-Carlo, Finite Differencing, Binomial Trees) produce by definition an approximation of the real “correct” result. In practice, we are forced to apply a numerical procedure to a class of problems where the accuracy of the procedure used within this class will vary from case to case. Therefore, when we speak about the accuracy of an algorithm, we are actually talking about the confidence level that we can have in an algorithm when it is applied to a class of problems. For example, we could say that for a given amount of computational resources we are more confident in the accuracy of the pricing of Asian options carried out with the Finite-Differencing approach than when Monte Carlo is applied.

More precisely, when we refer to the *accuracy* of an algorithm it should be understood in the following sense:

Two algorithms have the same accuracy if, having the parameters adjusted so that they run at equal speeds, produce results characterized by the same uncertainty interval.

Thus an algorithm is characterized by the degree of confidence (or certainty) in the approximations produced. That is, it can be characterized by the degree of confidence we have in its accuracy. The accuracy which an algorithm may have can vary from very high (for example, when we apply analytical formulae), to very low (for example, for unstable finite differencing algorithms).

The error within an algorithm is said to be controllable if we are able to eliminate (or

at least minimize) the error of an algorithm by carefully adjusting this parameters. In most instances errors are controllable, so the algorithms parameters can be adjusted in order to produce better results.

In some cases, the interval of uncertainty for a given confidence level can be explicitly estimated for a large class of problems (for example Monte Carlo pricing of option contracts). But in other cases the uncertainty interval could depend on either the parameter used or the particular problem considered (for example, in the case the finite differencing the accuracy of the result will depend on the payoff function, or equivalently the particular option contract considered).

Remark In many cases the speed of an algorithm is directly related to the accuracy of the algorithm. Within the Monte-Carlo and Finite Differencing as applied to options theory if the error of the algorithm is controllable then the accuracy of the algorithm is equivalent to the speed of the algorithm.

In the subsections below we describe in order of importance several factors related to our implementation of options pricing algorithms³ which can produce errors in the analysis of derivative securities.

5.7.1 Errors determined by algorithm instability

An algorithm is said to be unstable if it amplifies any other error source, for example floating point errors. Iterative algorithms are the most susceptible to instability as very small errors can grow at an exponential rate.

Finite Differencing

In Finite Differencing algorithms this error appears when the time step is too large relative to the price step. Instability errors are usually easily detectable as they tend to be very large, compared with other sources of errors. Each algorithm has its own threshold where it becomes unstable, in the case of finite differencing this is the ratio between time and price step. The explicit finite differencing algorithm is the most vulnerable to instability. The fully-implicit algorithm is much more robust but is not as accurate. Cranck-Nicholson is a compromise between robustness and accuracy.

Monte Carlo and Analytic Formulae

The best general algorithm when we refer to stability is, by far, Monte Carlo which is also the slowest. Even better is the analytical derivative pricing as applied opt the evaluation of European Vanilla options.

³There also exists a number of errors which can result from general computing issues such as rounding errors.

Remark Within the application of Finite Differencing and Monte Carlo to options theory it is almost always possible to eliminate instability by increasing the number of time steps.

5.7.2 Monte Carlo errors due to number of simulations used

Ideally in Monte Carlo you should perform an arbitrarily large number of simulations. Which we result in a confidence interval which is arbitrarily small. For a finite number of simulation the confidence level will have a definite positive size. Moreover, the confidence level generally will be larger than the correspond confidence interval for the finite differencing and analytic formulae. And hence the Monte Carlo procedure is slower (or less accurate) than either finite differencing or analytic formulae.

A significant advantage of the Monte Carlo procedure is that it does admit an explicit estimate of the error, or equivalently the confidence interval. For a confidence level $0 \leq c \leq 1$, the confidence interval is:

$$\mu - \frac{\text{erf}(c)\omega}{\sqrt{N}} < x < \mu + \frac{\text{erf}(c)\omega}{\sqrt{N}} \quad (5.7.29)$$

where,

- μ - is the average of all simulation results, or the returned value of the algorithm
- ω - is the standard deviation of the simulation results
- N - is the number of simulations
- erf - the error function

See the [Monte Carlo](#) section for further detail concerning the Monte Carlo procedure.

5.7.3 Finite Differencing errors due to wrong boundary conditions

In order to use finite differencing techniques to price derivative contracts you must set appropriate boundary conditions. As below in the section on boundary conditions within finite differencing, first you should try the general boundary condition, implemented by the `SecondOrderBoundaries` class. Even if this works for a large class of contracts, there is no guarantee that it will work in every situation. In fact, it always introduces a certain error in the final result.

Usually this error is small but there are circumstances when it can be significant. You can identify such an error by comparing the result of a finite differencing method with Monte Carlo (which does not require any boundary conditions, therefore is not affected by such an error).

The only way of eliminating such errors is by implementing the correct Dirichlet-type boundary conditions for the specific option contract you are pricing.

Remark In most cases boundary errors grow when the maturity time increases, so they can be spotted more easily by using contract with a long time until expiry, for example 5 or 10 years.

5.7.4 Finite Differencing errors due to the size of the price step

Solving a PDE numerically will always mean that some approximations will be used. In particular, the partial derivative of a function will be replaced by an approximation of it, using differences computed over the grid considered. Ideally the number of points within this grid should be arbitrarily large, or equivalently the grid should be arbitrarily dense. In reality though only a small number of points can be used due to computational restrictions. Again, comparison with Monte-Carlo can aid in identifying such errors.

Identify the step size error

In practice it can be quite difficult to discern between step size errors and errors resulting from the choice of boundary conditions. One difference between the two is that in many cases the former causes a shift of the prices for all maturities in one direction (while the boundary error is larger when the time to maturity increase). Hence, the algorithm will tend to underprice or overprice a given contract, no matter what maturity is used.

Controlling the Step size error

Fortunately the step size error is controllable. It can be minimized by increasing the number of points used to tabulate the price axis. If this number is too high compared to the number of time steps, the algorithms can become unstable. In particular, as the number of price steps increase the various implementations of the finite differencing will become unstable in the following order: explicit, Crank-Nicholson, fully implicit. Therefore, you may have to increase the number of time steps in order to maintain an algorithms stability.

5.7.5 Errors due to the size of the time step

This error is unlikely to generate a significant error in comparison to the stability errors when the finite differencing algorithms are applied. In the case of Monte Carlo which is very robust and hence in most cases stable, this error can have a significant influence. Saying this, such an error if it were to occur can be minimized by increasing the number of timesteps.

5.8 Greeks of Exotic Options

Recall that the Greeks describe the rate of change of an option's price with respect to various market variables. The Greeks can also be thought of as corresponding to various coefficients of the Taylor expansion of the price function with regard to the various market variables which it depends on. In short, the Greeks measure the sensitivity of an option (or portfolio of options) to various market variables in the following ways:

- **Delta** (Δ) - The rate of change of the option price to changes in the underlying asset price.
- **Theta** (Θ) - The rate of change of the option price to changes in the time remaining until the expiry of the option.
- **Gamma** (Γ) - The rate of change of the option delta, or equivalently the rate of change of the rate of change of the option price.
- **Vega** - The rate of change of the option price to changes in the volatility. Naturally, Vega will be measured with respect to the measure used within the option model under consideration.
- **Rho** - The rate of change of the option price to changes in the interest rates, in particular the risk free interest rate.

The definition of the Greeks in the case of Vanilla Options which were treated earlier and Exotic Options is identical. The differences lie in the way in which the Greeks in these two cases can be evaluated. In the case of Vanilla options via the Black-Scholes model we are able to derive analytic formulae for the price function and the Greeks (i.e. the derivative of the price function). In the case of Exotic options since the Black-Scholes model does not exhibit an analytic solution we are forced to use numerical procedures in order to estimate the price function and its corresponding derivatives (a.k.a. the Greeks). Within the previous sections we have discussed how the price function in accordance with the Black-Scholes model may be evaluated at a given point using one of the following generic approaches:

- Finite Differencing Techniques
- Monte-Carlo

Each of these pricing mechanisms can be used to estimate the Greeks of an Exotic Option and the corresponding values given for the Greeks may naturally differ. In the following sections we detail how we may estimate the value of the Greeks using these pricing mechanisms.

5.8.1 Finding Greeks of Exotic Options using Monte-Carlo

Evaluating one of the Greeks (excluding Gamma) is the equivalent to evaluating the derivative of the price function with respect to one of its variables. In the case of the Delta this variable is the underlying asset price, for vega the variable is the volatility and so on... In order to evaluate a given Greek which depends on the rate of change with respect to the variable η (say), at a given point in time $t = T$, we evaluate the price function:

$$f(T, \eta, \dots),$$

using Monte-Carlo approximation at the point $t = T$, $\eta = \eta(T)$, where $\dots$ corresponds to the other variables. We also evaluate the price function at the same point in time but where the η variable has been perturbed. That is, we set $\eta = \eta(T) + \epsilon$, where ϵ is a small positive number of modulus $\delta\eta$, and evaluate the function $f(T, \eta + \epsilon, \dots)$, using the Monte-Carlo approach. Then 'the Greek' which represents the rate of change with respect to the variable η , can be estimated by:

$$\frac{f(T, \eta(T), \dots) - f(T, \eta(T) + \epsilon, \dots)}{\delta\eta}$$

Remarks

- If in the above procedure η , is taken to be the underlying asset price then we will evaluate the Delta of the option, if η is taken to be the volatility then we will evaluate Vega, and so on...
- In order to minimize standard errors the number of trials used for the evaluation of the price function and its perturbation should be the same.

Evaluating the Gamma using Monte-Carlo

In order to evaluate the Gamma of an option using Monte-Carlo, you will need to evaluate the Delta using the above procedure at the evaluation point (i.e. $\Delta(T, \alpha(T), \dots)$), and for another point where the underlying asset's price has been perturbed (i.e. $\Delta(T, \alpha(T) + \epsilon, \dots)$). Then Gamma at the point $(T, \alpha(T), \dots)$, is given by:

$$\frac{\Delta(T, \alpha(T), \dots) - \Delta(T, \alpha(T) + \epsilon, \dots)}{\delta x}$$

where δx is the modulus of the perturbation ϵ .

5.9 Scenario Analysis

Scenario analysis allows for an asset (or portfolio of assets) to be stressed and for the resulting behavior to be analyzed. Such analysis is necessary within a Greek/VaR based market risk methodology to capture the risks due to outsized market moves. Though the standard

Greek/VAR⁴ methodology is adequate for the majority of market dynamics it generally will fail to reflect market risks due to shocks such as major geopolitical elements or market crashes. Examples of outsized market moves include the October 1987 stock market crash or the swings in the Treasury bond market in 1998 after the Russian government effectively defaulted on its national debt.

Applications to Risk Management and Trading

The major reason why the Greek/VaR methodology is insufficient lies in the fact that it under estimates the probability of outsized market moves. In theory such major outsized moves (for example of 3 standard deviation move in one day) will occur a few times every thousand years but in practice such outsized market moves have been observed to take place every ten years or so. That is, the historical distribution of assets prices has “fatter” tails than the tail implied from the normal distribution of asset prices which is one of the assumptions in the Black-Scholes model.

Though Scenario Analysis forms an important ingredient within most risk frameworks it is also a useful trading tool. Whether you wish to flatten out the exposure of a portfolio or take directional positions with particular characteristics a scenario grid representation will quickly communicate the directional exposure of the portfolio with respect to the market variables being considered. Moreover, if you wish to analyze the particular risk/reward profile of a particular trade before entering the position you can carrying out Scenario Testing on a number of scenario which you anticipate occurring. It this way you will know a prior the profit and loss implications of various differing scenario playing out. In this way you are will be able to construct an optimal trading approach according from your interpretation of the possible scenarios and there estimated probabilities.

What we will cover

Within this section we will consider the following two approaches to Scenario Analysis:

- Scenario Grid - The Profit/Loss of a portfolio after various shifts of two market variables from there present values. This gives a complete picture of the implications in terms of valuation (and hence profit and loss) of various market scenarios taking place.
- Scenario Testing - The Profit or Loss resulting from the a one off shift of all the market variables. Using this type of analysis you can analyze the effect on a portfolio if it underwent a similar major market shift to one which occurred in the past.

⁴The Value at Risk (or VaR) is a framework for producing a global risk parameter of a portfolio of assets. In particular, the n -day $m\%$ VaR of a portfolio of assets is the maximum expected lose with $m\%$ of confidence over a period of n days. The 5-day 95% VaR is the greatest loss expected over a 5-day period when the worst 5% of cases are excluded.

Remarks

- The VaR measure only captures information concerning the 99% (or 95%) least volatile days (at least in terms of the profit and loss). Hence, the VaR metric does not take into account the days which result in the most volatile portfolio value swings.
- The Greeks can be viewed as the leading coefficients of the Taylor expansion with respect to the market variables (price, interest rates, volatility, time etc) of the pricing function of the underlying asset. For small shifts in the market variables the leading terms of the Taylor expansion will offer a good approximation to the pricing function itself shifted. But as the shifts become larger to lower order terms will become less insignificant and sum of the leading terms of the Taylor expansion (i.e. the Greeks) will diverge from the pricing function itself. On the other hand, scenario analysis can cope equally well with large shifts in the underlying market variables since the pricing function (i.e. the entire Taylor expansion) is re-evaluated after the shift in the market variables has taken place.

5.9.1 Scenario Grids

A Scenario Grid with respect to two market variables either displayed as a grid of numerical values, or as a surface plot representing the values of a portfolio (or the resulting profit or loss) after each (scenario) shift, offers a quick and effective means to assess the type and degree of risk which a portfolio of assets is carrying.

Evaluation of the Absolute Scenario Grid

A scenario grid is a grid of numerical values, say with $\alpha \times \beta$ elements where the (a, b) entry is defined to represent the market value of the portfolio evaluated with the reference market variables. This value in practice is usually taken to be today's portfolio and market variables values.

Remark If we are considering the profit and loss resulting from various market moves then the value at a reference point (a, b) , will be zero.

The remaining points of the scenario grid are populated in the following fashion. Once the two market variables which are to be shifted and the degree to which each of these shifts will be (including whether the shift is a percentage or an absolute value) is decided the remaining scenario grid members can be evaluated as follow. The (x, y) element which is p , shifts in the first variable and q , shifts in the second variable is given by:

$$price(x + qs, y + pf)$$

where s (respectively f) are the number of shifts in the first (respectively second) market variable and $price$ represents the re-evaluation of the portfolio after the shifts to the two market variables whilst keeping the other market variables fixed. That is, the scenario

grid will display the absolute value of the portfolio under various shifts of the two market variables with respect to a reference value associated with a reference market condition.

Example of Absolute Scenario Grid

Below we give an example of a absolute Scenario grid where the initial value of the portfolio is 256,488, as stated in the grid element *D4*. The portfolio consider is made up of assets which depend on the price of the precious metal gold. The top axis represents corresponding shifts of the volatility of the underlying asset gold and the axis on the right hand side represent changes in the underlying price of gold. As you can see below the volatility and market price is shifted between -2 and $+2$, and the corresponding value of the gold portfolio is evaluated and the grid point is populated.

	A	B	C	D	E	F
1		-2%	-1%	0%	1%	2%
2	-2%	224,987	235,976	245,697	268,954	281,489
3	-1%	225,257	243,459	254,667	256,873	286,486
4	0%	225,478	245,879	256,488	265,489	287,456
5	1%	225,854	236,976	258,457	278,546	293,566
6	2%	226,548	240,589	256,485	286,547	305,487

Fig. The ‘Volatility vs Underlying’ Absolute Scenario Grid of the Gold Portfolio

Evaluation of Relative (or P/L Change) Scenario Grid

In the case of the relative (or P/L Change) scenario grid, where we consider the profit and loss implications of various market moves the scenario grid can be evaluated by just subtracting the value of the reference market value from the other values associated with various shifts in the market variables. That is, in the notion used above the (x, y) element within the relative (or P/L change) scenario grid is given by:

$$price(x + qs, y + pf) - R$$

where R is the reference market value of the portfolio being considered. In the gold portfolio example considered above the reference market value of 256,488 (see grid element *D4* above).

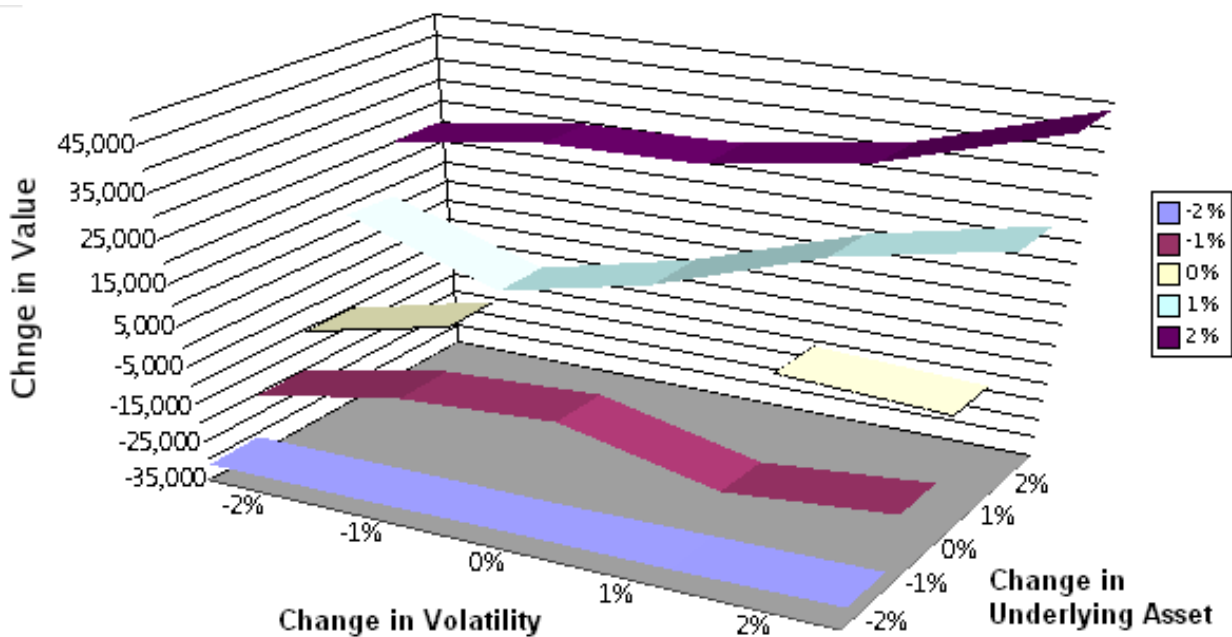
Example of Relative (or P/L Change) Scenario Grid

If we consider the example described above of the gold portfolio, which has a reference value on 256,488, and the underlying gold price and volatility is shifted between -2 and $+2$ then the relative (or P/L change) scenario grid is:

	A	B	C	D	E	F
1		-2%	-1%	0%	1%	2%
2	-2%	-31,501	-20,512	-10,791	12,466	25,001
3	-1%	-31,231	-13,029	-1,821	385	29,998
4	0%	-31,010	-10,609	0	9,001	30,968
5	1%	-30,634	-19,512	1,969	22,058	37,078
6	2%	-29,940	-15,899	-3	30,059	48,999

Fig. The 'Volatility vs Underlying' Relative Scenario Grid of the Gold Portfolio

If we plot these values within a 3D graph then the nature of the gold portfolio becomes clearer. In particular, from the graph below we can see that generally as the volatility of gold increases the value of the portfolio increases and this increase is amplified as the value of the underlying asset (in this case gold) increases. In short, a 3D graph of a scenario grid allows the nature of a portfolio against two market variables to be quickly and easily understood.



3D Graph of the 'Volatility vs Underlying' Relative Scenario Grid of the Gold Portfolio

5.9.2 Scenario Testing

Scenario Testing (also known as back testing) is another technique which allows the portfolio to be put under various stresses and to measure the implications in terms of the portfolio's profit and loss. Scenario Testing is generally applied in two instances:

- Trading Tool - Test a trading idea against various future scenarios.

- Risk Management (Back Testing) - Stress a portfolio according to past (extreme) market dynamics.

Mechanics of Scenario Testing

Whether you intend to use Scenario Testing as a trade or risk management tool to basic mechanics of its use are the same. The general procedure is to re-evaluate an asset (or portfolio) under a shift of all the market variables which the asset (or portfolio) depends on. For an asset this will involve performing the following steps:

- Identify the market variables which an asset depends. Since the Scenario Analysis depends on a pricing model this will involve going back to the pricing models being used for the given asset and identifying the models market variables.
- Shift the market variables according to the market scenario which you wish to analyze.
- Re-evaluate the asset according to the shifted market variables using the same model.

In order to undertake Scenario Testing of a Portfolio, we should perform to following:

- Identify all the market variables which the Portfolio is exposed to. Will we involve identifying the variables involved in the pricing models of the assets within the portfolio.
- Shift the market variables according to the market scenario which you wish to analyze.
- Re-evaluate the assets of the portfolio according to the shifted market variables.

By re-evaluating the asset (or portfolio) after a shift in the market variables has taken place we can see the effect on the value of an asset (or portfolio) which a given supposed market dynamics will have on the assets (or portfolios) value.

Using Scenario Testing as a Trading Tool

Once a trading idea has been identified and the possible future scenarios classified, the various ways in which to express a trading idea can be analyzed against the various future scenarios considered. By analyzing the possible scenarios against the original trading idea will help in assessing where to put stops and how to adjust the trading strategy as the trade evolves.

For example, say you take a directional position on the underlying asset price through a trading approach which hedges most of the volatility and interest rate risk at the point when the trade is entered. Further, you decide that you ideally do not wish to be exposed to excessive interest rate or volatility risk. Now, one of the envisaged scenarios plays out and from your a prior analyses you know that your exposure to volatility will grow larger than you feel happy with. At this point you hedge the volatility according to a predecided strategy, which you analysed aprior within your original senario analysis before entering

the trade. Hence, through Senario Testing you are able to build a robust trading approach which can manage the plan the whole trade life cycle under various senarios.

Using Senario (or Back) Testing within a Risk Management Framework

The application of Senario (or Back) Testing within a VaR/Greek risk methodology is particularly important since extreme market moves (i.e. the worst 1of 99stock amrket crash of October 1987, taken place significantly more often than would be expected from the lognormal distribution of asset prices.

Senario Testing within a risk management framework involves the following steps:

- Observe the market variables which a given portfolio being considered is exposed to.
- Record the historical market events for which the moves in the market variables which the portfolio under consideration is exposed to where particularly large.
- Re-evaluate the portfolio as if the extreme market variable shifts which occured in the past happened to the present value of the market variables.

By performing the above procedure the effect that an extreme historical market move would have on the considered portfolio if that market move would re-occur. In this way you can ascertain to what degree a given portfolio is “hedged” against extreme market moves. Since in times of market turmoil liquidity often dries up, and hence the hedging of risk during extreme market moves is usually impossible.

Chapter 6

Programmer's Guide

This chapter describes development techniques for various business solutions that make use of the functionality provided by our J2SE™ Component. We analyze both standard and enterprise Java solutions including stand-alone Java applications, Java Applets, JSP pages and Enterprise JavaBeans™.

6.1 Developing with J2SE

The J2SE Edition of this product offers core-level functionality to the Java developer allowing the implementation of fully personalized components and applications for specific business problems on a variety of deployment environments. Irrespective of the complexity of the project a programmer may be working on, this J2SE Component can be integrated in an equally straightforward manner by providing the needed functionality in a compact and precise way.

The Options and Futures v2.5 component comes wrapped inside a Java JAR file called OptionsJ2SE.jar. This class library contains all classes that provide the functionality offered by this J2SE Component as documented inside the API reference directory of this package (*/JavaDocs*). Depending on the type of Java solution you are developing, you will be using this JAR file in a specific way. Once you have chosen a particular implementation of a deployment framework, the structure of your source code will need to adapt accordingly. In the following discussion we describe several basic J2SE implementation examples that can be used to make use of our product's functionality.

6.1.1 Stand-alone Java Applications

A stand-alone Java application is a Java class which acts as a client for any type of Java components ranging from JavaBeans, class libraries and EJB components. The source code listed below defines a standard stand-alone application skeleton which communicates with a class within a J2SE Component. By creating an instance, calling a method, sending in a set of parameters and then retrieving the returned result of the method call. As soon as the method call has gone through successfully the method continues by displaying its

result on the screen.

```
/*
 * The class we are using is located in the webcab.lib.j2sepackage package.
 */
import webcab.lib.j2sepackage.*;

public class StandAloneExample {

    public static void main (String[] args) {

        /*
         * Creates an instance of the J2SEClass class
         */
        J2SEClass instance = new J2SEClass ();

        /*
         * Defines the method parameter.
         */
        double parameter = 25;

        /*
         * Invokes a method with the specified parameter
         * and puts the result in the result variable.
         */
        double result = instance.computeResult (parameter);

        /*
         * Prints out the retrieved result.
         */
        System.out.println ("Method 'computeResult' in class J2SEClass
            returned " + result + ".");
    }
}
```

If the method `computeResult` of the `J2SEClass` class returned 100, this stand-alone application would print the following:

```
Method 'computeResult' in class J2SEClass returned 100.
```

6.1.2 Java Applets

Applets run within a Java enabled web container allowing real-time user interaction on the Internet. The source code listed below defines an Applet which performs the same calculations as the stand-alone example above and then graphically displays the result.

```
/*
 * The class we are using is located in the webcab.lib.j2sepackage package,
 * while the Applet specific classes reside in javax.swing.
 */
import webcab.lib.j2sepackage.*;
import javax.swing.*;

public class AppletExample extends JApplet {

    public void init () {

        /*
         * Creates an instance of the J2SEClass class
         */
        J2SEClass instance = new J2SEClass ();

        /*
         * Defines the method parameter.
         */
        double parameter = 25;

        /*
         * Invokes a method with a specified parameter
         * and puts the result in the result variable.
         */
        double result = instance.computeResult (parameter);

        /*
         * Prints out the retrieved result.
         */
        getContentPane ().add (new JLabel ("Method 'computeResult' in
            class J2SEClass returned " + result + "."),
            java.awt.BorderLayout.CENTER);
    }
}
```

This Applet source code will be compiled and placed together with all classes from the OptionsJ2SE.jar file in another Java JAR file and be deployed inside a web container. Users connecting through a web browser will first download this newly created Java JAR file and then interact with the Applet's graphical interface.

6.2 J2EE™ Solutions Using J2SE Components

The Java™2 Enterprise Edition platform provides a great variety of business framework solutions, which transfer hard-coding to the Application Server providers and enables the developer to exclusively concentrate on the business problem. In the following sections we discuss two of the most popular J2EE technologies, Java Server Pages (JSP) and Enterprise JavaBeans (EJB), which are universally supported by all Application Server providers.

6.2.1 Writing JSP Pages

JSP Pages are very much like Applets since they both basically offer the same functionality to the end-user, interaction within a web page. But even though they both display their results in a browser, the main difference is the fact that JSP pages are server-side applications, which never get downloaded onto the client's machine and never use local CPU or memory resources. Yet as previously stated the code you will write when designing your JSP pages to use our J2SE Components will be very similar to client side applications we have already described. That is, you will need to make the following steps:

- Create an instance of the class you wish to use
- Make a method call by sending the required parameters
- Display the result in HTML form

The following JSP page exemplifies this process:

```
<!-- The class we are using is located in the webcab.lib.j2sepackage package.
-->
<%@page import="webcab.lib.j2sepackage.*"%>
<HTML>
<HEAD><TITLE>JSP using J2SE Component Example</TITLE></HEAD>
<BODY>
<%
    /*
     * Creates an instance of the J2SEClass class
     */
    J2SEClass instance = new J2SEClass ();

    /*
     * Defines the method parameter.
     */
    double parameter = 25;

    /*
     * Invokes a method with the specified parameter
     * and puts the result in the result variable.
     */
    double result = instance.computeResult (parameter);
%>
<!-- The JSP page displays the result inside the web browser. -->
Method 'computeResult' in class J2SEClass returned <%=result%>.
</BODY>
</HTML>
```

The JSP page will be compiled into an HTML page by the web container on the server and be served to the client's machine where the end-user will be able to read the computed result, as if it had been run locally.

6.2.2 Designing EJB™ Components with J2SE Components

Enterprise JavaBeans™ are server-side components which provide the same business functionality as J2SE Components while not taking up local system resources. Developers writing EJB components that use the functionality provided by J2SE Components will need to include the OptionsJ2SE.jar file with their enterprise deployment archive (EAR) and perform local calls like shown in the previous examples. The following source code defines the implementation class of a stateless Session Bean which makes calls to a J2SE class.

```
/*
 * The class we are using is located in the webcab.lib.j2sepackage package,
 * while the EJB specific classes reside in javax.ejb.
 */
import webcab.lib.j2sepackage.*;
import javax.ejb.*;

public class EJBClassExample implements SessionBean {

    /*
     * EJB specific methods.
     */
    public void ejbActivate() throws RemoteException {}
    public void ejbPassivate() throws RemoteException {}
    public void ejbRemove() throws RemoteException {}
    public void setSessionContext (SessionContext c)
        throws RemoteException {}

    /*
     * Business enterprise method.
     */
    public boolean checkResult (double parameter) {

        /*
         * Creates an instance of the J2SEClass class
         */
        J2SEClass instance = new J2SEClass ();

        /*
         * Invokes a method with the received parameter
         * and puts the result in the result variable.
         */
        double result = instance.computeResult (parameter);

        /*
         * Returns true if the result is positive, false otherwise.
         */
        return (result >= 0);
    }
}
```

Together with its home and remote interfaces this class will encapsulate a stateless Session EJB that serves client-side requests by making local J2SE calls.

Remark For further details concerning the use of this component as an EJB, please see the Programmers Guide of the J2EE Editions PDF documentation.

6.3 The Options Module Methods

The Options module consists of the following eleven .NET components:

1. `EuropeanEvaluation` - evaluates European call and put options on stocks, indexes, currencies and futures using the Black-Scholes formulae
2. `EuropeanDelta` - calculates the delta of European options on stocks, indexes, currencies and futures
3. `EuropeanGamma` - calculates the gamma of European options on stocks, indexes, currencies and futures
4. `EuropeanTheta` - calculates the theta of European options on stocks, indexes, currencies and futures
5. `EuropeanVega` - calculates the vega of European options on stocks, indexes, currencies and futures
6. `EuropeanRho` - calculates the rho of European options on stocks, indexes, currencies and futures
7. `OptionStrategies` - considers the payoff functions of different option trading strategies, in particular for spreads and combinations
8. `BinaryOptions` - considers the Binary (or digital) options
9. `PutCallParity` - within this component we implement put-call parity relations for Vanilla and Binary Options where the underlying asset does not pay dividends and for Vanilla options where the underlying asset pays a continuous dividend
10. `Volatility` - consists of a collection of methods for estimating the value of the volatility from historical data using direct calculation, ARCH, EWMA and GARCH models
11. `ImpliedVolatility` - calculates the implied volatility for call and put options for Black-Scholes formulae

6.4 The Futures Module Methods

The Futures module consists of the following six .NET components:

1. DailyReporting - we consider methods related to the daily reporting and management of a futures trading account. This includes margin requirements, daily P&L, total equity and excess margin.
2. Forwards - we consider evaluation of forward contracts between two parties.
3. FuturesEvaluation - we consider evaluation of future contracts on stocks, bonds and indexes.
4. FuturesHedging - we consider the use the futures contracts for the hedging of risks within stock, commodity, bonds and currency markets.
5. FuturesOnCommodities - we consider the modeling of futures contracts on investment and consumption commodities.
6. Interest - we consider basic methods for the evaluation of interest bearing investments.

6.5 The Exotic Options Module

The Exotic Options module within the WebCab Options and Futures J2SE Component provides methods for the evaluation of exotic options. In particular, this module is able to effectively evaluated multi-asset European options, American options, Lookback Options, Asian Options and Bermuda Options. These options are priced according the Black-Scholes-Merton formulation using Monte-Carlo and finite difference techniques.

6.6 Specifying Contract Details

6.6.1 Payoff Function

As seen in the mathematical documentation, the most important thing that differentiates option contracts is the payoff function. In the most general form, a Bermudan strongly path-dependent option has a payoff function of the following form:

$$\text{payoff} = P(S, I, t) \tag{6.6.1}$$

where S is the price, I is the path dependent value and t is the time. In case we have more than one asset, S and I are treated as arrays under Java. In the case where the payoff function depends on more than one asset the variables S and I , will become vectors.

Examples: In the case of both Asian and Loopback options the values given for S (see below), will correspond to the market price of the underlying asset at some time T , on which the option is defined. The path dependent variables will depend on the way in which the option is defined. In particular, for Asian and Loopback option we have:

- **Asian** - $I =$ 'Average of the price over a specified period evaluated at a given point in time'
- **Loopback** - $I =$ 'Maximum of the price over a specified period evaluated at a given point in time'

6.6.2 The Payoff Function Interface

Before calling any pricing methods, you must set the payoff function. In order to do this you must implement the `PayoffFunction` interface. This interface contains the method:

```
getValueAt(double t, double[] x, double[] path_dependent_values, int n)
```

Your `getValueAt` method must return the value of the payoff function, given the vector of current asset prices (x) and the vector of path dependent values; t is the current time and n is the total number of assets. Both vectors have n elements. We include with our product implementations of the `PayoffFunction` interface for some common types of options. The following is a list of option types for which a payoff function is supplied together with the name of the class.

- **Options with only one asset**
 - Simple, vanilla options - `SimplePayoff`
 - Binary options - `BinaryPayoff`
 - Standard strongly path dependent options - `StronglyPathDependentPayoff`
- **Multi-asset options**
 - Spread strategies - `SpreadStrategyPayoff`
 - Strangle strategies - `StrangleStrategyPayoff`
 - Straddle strategies - `StraddleStrategyPayoff`

6.6.3 Boundary conditions

Boundary conditions are required only when using finite differencing methods, namely the `FiniteDifference` bean. They can become quite complex if used for multi-asset options and even for single-asset they are rather tricky to set in order to express the true financial behavior of option values. The most common types of boundary conditions are:

- Dirichlet conditions - you supply the value of the function on the boundary.
- Neumann conditions - you supply the first derivative of the function on the boundary.

We give the user the choice of supplying its own boundary condition of Dirichlet type. The interface `DirichletBoundaries` was designed for this purpose. It contains the method:

```
getValueAt(double t, int i, boolean i_max, double[] not_fixed_x, int m)
```

The question is how this method should be implemented. Think, for example, that we have a one-asset option to be priced. In this case the PDE will have two variables: t (time) and S (price). The final condition is clearly a function of one variable: S . The boundary condition is more complex. Boundaries are always supplied in pairs. We have a limit for low values and another limit for high values. This is due to the fact that there is no preferred direction on the price axis (unlike the time axis). So that's the utility of the i_max parameter. If i_max is `true` then the method must return the value from the boundary with the highest price. If it is `false` the method will return the value from the lowest price boundary. Each boundary is a function of time.

Now, if the option is multi-asset, things get complicated even more. Here you will see the meaning of the last two unexplained parameters: i and not_fixed_x . Suppose we have a three-asset option. There will be then three pairs of boundaries, each corresponding to one of the asset prices. But here is the big difference: the boundaries are no more functions of time, but are now functions of time and the two remaining asset prices. So we fix one asset price on its highest or lowest value and we must supply the option value at any time and for any combination of the other two asset prices. Parameter i indicates the asset with fixed price (counting begins from 0). Parameter not_fixed_x is a vector of $m - 1$ elements - the prices of the remaining assets. t is, of course, the time and m is the total number of assets.

The interface `DirichletBoundaries` extends a more general interface, `Boundaries`, which has only one method: `int type()`. This method informs the algorithm about the boundary type. In the current implementation only Dirichlet boundaries can be supplied by the user, so, when implementing the `DirichletBoundaries` interface, the method `type` must return the predefined constant `BT_DIRICHLET`. To avoid the task of always writing this method, an abstract class, named `Dirichlet`, which provides `type` and leaves `getValueAt` unimplemented, was included. You simply have to extend this class instead of implementing the interface.

The class `SimpleBoundaries` which extends `Dirichlet` can be used for (single) asset vanilla put and call options.

There is another class, `SecondOrderBoundaries`, included in the package. This is no longer a Dirichlet boundary. The following assumptions are made:

- At $S = 0$ (the low boundary), for most contracts (including calls and puts) the diffusion and drift terms *switch off*. This means that on $S = 0$ the payoff is guaranteed,

resulting in the condition:

$$\frac{\partial V}{\partial t}(0, t) - rV(0, t) = 0 \quad (6.6.2)$$

- At $S \rightarrow \infty$ (the high boundary), the payoff is at most linear in the underlying, so the condition is:

$$\frac{\partial^2 V}{\partial S^2}(S, t) \rightarrow 0 \text{ as } S \rightarrow \infty \quad (6.6.3)$$

The advantage of `SecondOrderBoundary` is that it works well with a broad range of options. It is therefore a good starting point if you find it hard to supply a correct Dirichlet boundary.

6.7 Support for Developers

6.7.1 Compilation and Custom Modification of Source Code

This J2SE Components source files have been compiled using Sun's J2SDK1.4.2, should you prefer compilation of the source code using another compiler then please contact us. Some functions (for example standard mathematical operations) are embedded within the source code, if you license classes which you believe offer a similar level of functionality from a third party and wish to include these classes within our components then please contact us. Both of the above services are offered free of charge at WebCab's discretion.

6.7.2 Online

Should you have any questions or queries, please feel free to contact us via our support forum at:

<http://www.webcabcomponents.com/support/index.php>

For updates and new releases, visit:

<http://www.WebCabComponents.com/Java/index.shtml>

Chapter 7

Examples

Within this chapter we detail the examples provided with this J2SE Component. We provide with this Component examples of two types:

1. **QA Examples**
2. **Custom Examples**

7.1 Question and Answer (QA) Client Examples

7.1.1 Overview

Within this folder we offer Java client examples for the J2SE Business Components provided within this package. In particular, for each business method contained within each business Component we provide a client side example; which illustrates how functionality contained within this component can be applied to solve real world problems. In addition, to these examples we also include custom applications which solve some of the generic problems which this Component is designed to address.

7.1.2 Structure of QA Examples Directory

By drilling down the QA Client directory structur you will find the following six directory levels:

1. Client Level
2. Technology Level
3. Business Module Level
4. Business Class Level
5. Example Level
6. Custom Example Level

which have the structure as shown in the following diagrams.

7.1.3 Top Four levels of the QA Directory Structure

```

      (Technology Level)  (Business Module Level)          (Business Class Level)

Client +
      |
      + Java -----+
      |              |
      + Readme.txt   + 'Business Module 1' -----+
                      |                             |
                      + 'Business Module 2' --+... + 'Business Class 1' -----
                      |                             |
                      + QALibrary.jar           + 'Business Class 2' --+...
                                                  |
                                                  + ...

```

7.1.4 Bottom Two levels of the QA Directory Structure

```

      (Example Level)          (Custom Example Level)

-----+
      |
      + 'BusinessClass'.java
      |
      + compile.cmd (.sh for Linux)
      |
      + run.cmd (.sh for Linux)
      |
      + run_gui.cmd (.sh for Linux)
      |
      + 'CustomClient' -----+ 'CustomClient'.java
      |                       |
      + ...                   + compile.cmd (.sh for Linux)
                              |
                              + run.cmd (.sh for Linux)

```

7.1.5 Quick Start Guide

Browse down to the particular client within the business module for the given client technology you are interested in. Run the compile.cmd (.sh for Linux), script in order to compile the example and then run the run.cmd (.sh for Linux), script in order to run the example.

7.1.6 Explanation of the QA Directory Structure and its files

1. Client Level

The directory containing the Questions and Answer Examples is named 'QAClients'. This directory can be located by using the Index.html file, which is presented at the end of the installation process, by selecting:

Run Examples > Client Examples Directory

Within this folder you will also find the following file:

- Readme.txt - this readme file

2. Client Technology Level

Within the 'Client' folder is the Technology Level of the QA directory structure. Within this folder you will find sub-directories; which contain examples written in each of the Java Client Technologies in which the clients have been implemented.

As mentioned above within the overview we provide an implementation of each 'Question and Answer (QA)' example using each of the client technologies used. Selecting one of these folders in order to view the client implemented within the corresponding technology.

3. Business Module Level

After selecting one of the technology sub-folders at the 'Client Technology Level' (see (2) above) you will enter a folder which contains a sub-folder for each of the modules contained within this component package. The modules are convenient collections of business classes containing the business functionality offered by this component. Each module has a sub-folder which contains the 'Question and Answer' examples of each method of the classes which it contains.

This directory will also contain the following file:

- QALibrary.jar - contains utility functionality necessary for running the clients.

4. Business Class Level

At the business class level under the module level we are presented with one sub-folder for each of the classes within the business module. Each of these folders contains the files of the examples for each of the methods contained with the respective class.

5. Example Level

At the Example level you will be presented with the files which allow you to compile

and then run the examples of the application of the Business methods of the respective class. The files within this directory which constitute the Client example are as follows:

- Source Code File(s) - Depending on the Java compatible technology selected at the 'Technology Level' this folder will contain a source code file(s) of the client.
- Compile Batch File - Assuming that you have access to the relevant compiler the compile batch file (compile.cmd or compile.sh) once run will compile the client example from the folders source code file. Below we include some remarks on how to obtaining suitable Java compiler in order to be able to compile and then run the Java examples.
- Run Batch File - The run batch file (run.cmd or run.sh) will run the examples executable file (exe) which is generated from the compilation of the source code file. If the example over runs your console screenful then press space in order to page down or enter is order to scroll down.

6. Custom Example Level

At the Custom Example level you will find a source code file containing the implementation of the Application which addresses a typical question for which the Component is designed. The source code has a linear structure with full inline commentary. By just running the compile and then run scripts contained within this directory you will be able to solve to given problems view the results obtained. The files contained within each Custom Client Example directory are as follows:

- Source Code File(s) - The custom clients source code in the appropriate client technology.
- Compile Batch File - Assuming that you have access to the relevant compiler the compile batch file (compile.cmd or compile.sh) once run will compile the client example from the folders source code file.
- Run Batch File - The run batch file (run.cmd or run.sh) will run the examples executable file which is generated from the compilation of the source code file. If the example over runs your console screenful then press space in order to page down or enter in order to scroll down..

7.1.7 Remarks on Java compilers

1. Required compilers and how to test for them

In order to compile the Java examples you will require a suitable Java compiler which should be installed and configured within your environment variable PATH. The easiest way in which to test whether you have access to a Java compiler at your command prompt is to type the follow command at the command line:

java - Invokes the Java compiler

2. How to obtain the required Java compiler

Compilers for the Java programming language are provided within Sun's Java SDK which can be obtained from:

<http://java.sun.com>

7.2 Options Custom Examples

We provide examples for each of the methods provided by this module.

7.2.1 Volatility

Within this subsection we describe the examples provided within the:

```
VolatilityClient.java
```

client which is contained within the Client\Volatility folder of this package. Within this client we provide one example for each of the methods within the `Volatility` class.

Example 1: `volatilityYearsDaysRescaling(double volatilityPerAnnum, double daysRescaledTo, int convention)`

Problem: If the volatility measured over a year (i.e. 365 days) is 12%, what is same assets volatility wrt 5 day period?

Solution: The volatility with respect to 5 days is given by:

```
volatilityYearsDaysRescaling(0.12,5);
```

from the `Volatility` class.

Example 2: `volatilityHistoricalEstimate(double[] assetPrices)`

Problem: What is the historical estimate of the volatility of an asset which over the past 12 months, took the following values:

[23.4, 24.1, 24.5, 23.9, 24.5, 24.9, 25.0, 24.8, 25.1, 25.7, 26.0, 25.8]

Solution:

Example 3: `volatilityHistoricalEstimateWithDividends(double[] assetPrice, double[] dividendsPaid)`

Problem: What is the historical estimate of the volatility of an asset which over the past 12 months, which took the following values:

[23.4, 24.1, 24.5, 23.9, 24.5, 24.9, 25.0, 24.8, 25.1, 25.7, 26.0, 25.8]

and during this period paid a dividend of 2 in the 1st and 7th month?

Solution:

Example 4: `standardErrorHistoricalVolatilityEstimate(double numberOfDays, double volatilityEstimate)`

Problem: Given an error estimate of the approximation provided in Example 2 and 3.

Solution: In order to estimate the error we use a method with the volatility class which approximates the standard error.

Example 5: `archVolatilityEstimate(double longTermVolatility, double weightOfVolatility, double[] observations, double[] weights)`

Problem: If the long term volatility of the S&P500 index is 1.32% per day and the value of the index over the last week has been: 1147.39, 1144.58, 1138.49, 1131.87, 1148.7. According to the ARCH Volatility model what will the volatility be?

Solution: In order to apply the ARCH model we are required to select weights for the historical values of the index and the long term average of the volatility. The weights selected will depend upon the personal judgment of the user. If we selected the weights: 0.39, 0.20, 0.15, 0.10, 0.05 for the last week, and a weight of 0.11 for the long term volatility. Now the parameters in the ARCH model are:

- long term volatility - 0.0132
- long term volatility weight - 0.11
- observations array - 1147.39, 1144.58, 1138.49, 1131.87, 1148.7
- weights array - 0.39, 0.20, 0.15, 0.10, 0.05

By call the `archVolatilityEstimate` method in the following sense:

```
double[] observations = new double[5];
double[] weights = new double[5];
double result = archVolatilityEstimate(observations, weights);
System.out.println(result);
```

We are able to conclude that the ARCH estimate of the volatility is XX.

Example 6: `ewmaVolatilityEstimate(double weightRatio, double i1thDayEstimateOfVolatility)`

Problem: What is the estimate of the volatility of the price today according to the Exponentially Weighted Moving Average model when the price volatility yesterday was recorded to be 1% per day, the asset price at the start of trading yesterday was 20, and closed at 20.404? We take the weight ratio to be 0.90.

Solution: We calculate the volatility on day n using the EWMA model:

- The parameters are:
 - weight ratio - 0.90
 - (n-1)-th day estimate of volatility - 0.01
 - n-th day asset price - 20.404
 - (n-1)-th day asset price - 20
- The source code:

```
double result = ewmaVolatilityEstimate(
    0.90, 0.01, 20.404, 20);
System.out.println(result);
```

- The result: 0.011401523854261015
- Conclusion: The estimate of the volatility today is 1.14%

Example 7: `ewmaVolatilityEstimateInduction(double weightRatio, double first-DaysVolatilityEstimate, double[] endOfDay, double[] startOfDay)`

Problem: What is the estimate of the price volatility today according to the Exponentially Weighted Moving Average model when the weight ratio (for the past 5 days) is taken to be 0.90, the volatility 5 days ago was 1% per day, and the asset price at the start and end of the last 5 trading sessions was:

```
startOfDay = [21, 21.3, 21.2, 21.4, 21.5]
endOfDay = [21. ]
```

Solution:

Estimating the volatility with GARCH(1,1) model

Problem: Suppose that the estimate of the volatility on day $n - 1$ is 1.6% per day and has a weight of 0.86, the proportional change in the market variable on day $n - 1$ is 1% with a weight of 0.13 and the long term variance is 0.0002 with a weight of 0.01.

Solution: We calculate the volatility on day n using the GARCH(1,1) model:

- The parameters are:
 - long term variance - 0.0002
 - long term variance weight - 0.01
 - (n-1)-th day volatility estimate - 0.016
 - (n-1)-th day volatility estimate weight - 0.86
 - continuously compounded return during n-th day - 0.01
 - asset price weight - 0.13
- The source code:

```
double result = garchVolatilityEstimate(  
    0.0002,0.01,0.016,0.86,0.01,0.13);  
System.out.println(result);
```

- The result: 0.015334927453366056
- Conclusion: The volatility in the n-th day is 1.53%.

7.2.2 ImpliedVolatility

Problem: Suppose that the value of a non-dividend paying stock is 1.875 when the stock price is 21, the strike is 20, the risk-free interest rate is 0.1, and the time to maturity is 0.25 (three months).

Solution: We calculate the implied volatility for a call option on a non-dividend paying stock:

- The parameters are:
 - stock price - 21
 - strike - 20
 - risk-free interest rate - 0.1
 - time to maturity - 0.25
 - option value - 1.875

- The source code:

```
double result = callVolatility(21,20,0.1,0.25,1.875);  
System.out.println(result);
```

- The output: 0.23451260510754787
- Conclusion: The implied volatility is approximately 0.2345, or 23.45% per annum.

7.2.3 EuropeanDelta

Problem: A bank has written a six-month European option to sell £1,000,000 at an exchange rate of 1.6000. Suppose that the current exchange rate is 1.6200, the risk free interest rate in the United Kingdom is 13% per annum, the risk-free interest rate in the United States is 10% per annum, and the volatility of sterling is 15%.

Solution: We calculate the delta for the put option on a currency:

- The parameters are:
 - foreign interest rate - 0.13
 - exchange rate - 1.620
 - strike - 1.600
 - risk free interest rate - 0.1
 - volatility - 0.15
 - time to maturity - 0.5

- The source code:

```
double result = putDeltaOnCurrency(0.13, 1.620, 1.600, 0.1, 0.15, 0.5);  
System.out.println(result);
```

- The output: -0.457793999252818
- Conclusion: The delta for this European put is approximately -0.458.
It means that for small exchange rate changes the price of the put goes down by 45.8% of the increase in the value of the currency.

7.2.4 EuropeanTheta

Problem: Consider a four-month European put option on a stock index. The current value of the stock index is 305, the strike price is 300, the average weighted dividend yield of its constituents is 3% per annum, the risk free interest rate is 8% per annum, and the

volatility of the index is 25% per annum. What is the theta of this option?

Solution: We use the method `putThetaWithYield` for this index option:

- The parameters are:
 - yield - 0.03
 - index value - 305
 - strike - 300
 - risk-free interest rate - 0.08
 - volatility - 0.25
 - time to maturity - 1.0/3

- The source code:

```
double result = putThetaWithYield(0.03, 305, 300, 0.08, 0.25, 1.0/3); System.out.println(result);
```

- The output: -18.152807990840586
- Conclusion: The theta on this European put option is approximately -18.15 per annum. When time is measured in days theta is $-18.15/252 = -0.072$.

7.2.5 EuropeanGamma

Problem: Consider a four-month European put option on a stock index. The current value of the stock index is 305, the strike price is 300, the average weighted dividend yield of its constituents is 3% per annum, the risk free interest rate is 8% per annum, and the volatility of the index is 25% per annum. What is the gamma of this option?

Solution: We use the method `optionGammaOnIndex` of this put option.

- The parameters are:
 - yield - 0.03
 - index value - 305
 - strike - 300
 - risk-free interest rate - 0.08
 - volatility - 0.25
 - time to maturity - 1.0/3

- The source code:

```
double result = optionGammaOnIndex(0.03,305,300,0.08,0.25,1.0/3); System.out.println(result);
```

- The output: 0.008571613944348518
- Conclusion: The gamma for this European put option is approximately 0.00857. Thus an increase of 1.0 (from 305 to 306) in the index increases the delta of the option by approximately 0.00857.

7.2.6 EuropeanVega

Problem: Consider a four-month European put option on a stock index. The current value of the stock index is 305, the strike price is 300, the average weighted dividend yield of its constituents is 3% per annum, the risk free interest rate is 8% per annum, and the volatility of the index is 25% per annum. What is the vega of this option?

Solution: We calculate the vega of this put option on a index using the method `optionVegaOnIndex`

- The parameters are:
 - yield - 0.03
 - index value - 305
 - strike - 300
 - risk-free interest rate - 0.08
 - volatility - 0.25
 - time to maturity - 1.0/3

- The source code:

```
double result = optionVegaOnIndex(0.03,305,300,0.08,0.25,1.0/3); System.out.println(result);
```

- The output: 66.44786213550597
- Conclusion: The vega for this put option is approximately 66.44. A 1% or 0.01 increase in volatility (from 25% to 26%) increases the value of the option by approximately 0.6644 (0.01×66.44).

7.2.7 EuropeanRho

Problem: Consider a four-month European put option on a stock which does not pay a dividend. The current value of the stock is 305, the strike price is 300, the dividend yield is 3% per annum, the risk free interest rate is 8% per annum, and the volatility of the stock index is 25% per annum. What is the rho of this option?

Solution: We calculate the rho of this put option on a stock using the `putRho` method.

- The parameters are:
 - index value - 305
 - strike - 300
 - risk-free interest rate - 0.08
 - volatility - 0.25
 - time to maturity - 1.0/3

- The source code:

```
double result = putRho(305,300,0.08,0.25,1.0/3); System.out.println(result);
```

- The output: -39.93790052033336
- Conclusion: The rho for this put option is approximately -39.9379. For a one-percentage-point or 0.01 increase in the risk-free interest rate (from 8% to 9%), the value of the option decreases by 0.3994 (0.01×39.94).

7.2.8 OptionStrategies

Payoff from a bull spread

Problem: A trader buys a call with a strike price of \$30 and sells a call with a strike price of \$35. The stock price at expiry is \$32. What is the payoff at expiry from this spread trade?

Solution: In order to evaluate the spread trade we use the `payOffBullSpread` method.

- The parameters are:
 - expiry price - 32
 - bought call strike - 30
 - sold call strike - 35
- The source code:

```
double result = payOffBullSpread(32,30,35);  
System.out.println(result);
```


- The output: 2.0
- Conclusion: The payoff for this bull spread is \$2.

Remark The dealing costs are not taken into account.

Payoff from a bear spread

Problem: A trader buys a call with a strike price of \$35 and sells a call with a strike price of \$30. The stock price at expiry is \$32. What is the payoff from the bear spread trade?

Solution: To calculate the payoff from this bear spread trade we use the `payOffBearSpread` method.

- The parameters are:
 - expiry price - 32
 - brought call strike - 35
 - sold call strike - 30

- The source code:

```
double result = payOffBearSpread(32,35,30);  
System.out.println(result);
```

- The output: -2.0
- Conclusion: The payoff for this bear spread is -\$2.

Payoff from butterfly spread

Problem: Consider a trader who feels that a significant price move in the next six months is unlikely. The trader buys a call with a 55\$ strike price, buys a call with a 65\$ strike price, and sells two calls with a 60\$ strike price. Suppose that the stock price is 61\$ at the expiry of the options. What is the payoff from this trade?

Solution: We calculate the payoff from this butterfly spread using the `payOffButterflySpread` method.

- The parameters are:
 - expiry price - 61
 - high brought call - 65
 - low brought call - 55
 - sold calls - 60

- The source code:

```
double result = payOffButterflySpread(61,65,55,60);  
System.out.println(result);
```

- The output: 4.0
- Conclusion: The payoff from this butterfly spread is \$4.

Payoff from strangle combination

Problem: A trader buys a call option with a strike price of \$50 and a put option with a strike price of \$45. Suppose that the stock price is \$47 at the expiry of the options. What is the payoff from this trade?

Solution: We calculate the payoff from this strangle combination using the method `payOffStrangleCombination`.

- The parameters are:
 - expiry price - 47
 - call strike -50
 - put strike - 45

- The source code:

```
double result = payOffStrangleCombination(47,50,45);  
System.out.println(result);
```

- The output: 0.0
- Conclusion: There is no payoff (positive or negative) from this strangle combination.

Payoff from straddle combination

Problem: A trader buys a call option and a put option, both with a strike price of \$100. At the expiry of the options the stock price is \$87. What is the payoff from this straddle combination?

Solution: We calculate the payoff from this straddle combination using the `payOffStraddleCombination` method.

- The parameters are:
 - expiry price - 87

- strike -100
- The source code:

```
double result = payOffStraddleCombination(87,100);  
System.out.println(result);
```

- The output: 13
- Conclusion: The payoff from this straddle combination is 13\$.

7.2.9 Put-Call Parity

Problem: Suppose that the stock price is \$31, the exercise price is \$30, the risk-free interest rate is 10% per annum and the price of a three month European call option is \$3. What is the price according to non-arbitration principles of the corresponding put option?

Solution: We calculate the price of the put option with the same strike and expiration date:

- The parameters are:
 - value of call - 3
 - stock price - 31
 - strike - 30
 - risk-free interest rate - 0.1
 - expiry time 0.25

- The source code:

```
double result = putEuropean(3,31,30,0.1,0.25);  
System.out.println(result);
```

- The output: 1.2592973608499776
- Conclusion: The value of the corresponding put is approximately \$1.26.

7.2.10 Binary Options

Problem: Suppose that the stock price is \$30 at expiry of a binary call option with strike 20 which has a payout of \$1. What is the payoff from the Binary call option?

Solution: We calculate the payoff of the binary call option using the method `payOffBinaryCall`.

- The parameters are:
 - expiry price - 30
 - strike - 20
 - payout - 1

- The source code:

```
double result = payOffBinaryCall(30,20,1); System.out.println(result);
```

- The output: 1.0
- Conclusion: The payoff from this binary call option is \$1.

7.2.11 Database Example with JDBC Mediator

The Database Example is located inside the *Client/Options/DatabaseExample* directory. The following steps are required before running the Database example.

1. Installing our Tables (MySQL Only)

In order to create the database structure from which you are able to load the output results, you will need to run the 'create.sql' scripts in the 'Data' subdirectory. Open the MySQL prompt from the 'Data' subdirectory and:

- Select (or create and then select) a database you can spare for tables named AMD, BEAS, DELL, IBM, MSFT, ORCL, and RATE. For example, if you have decided using the 'test' database, you would optionally type the SQL command to create it (unless it already exists):

```
CREATE DATABASE test;
```

and then, you would type the following to select it:

```
USE DATABASE
```

- Run the 'create.sql' SQL script located in the 'Data' subdirectory of the current directory by typing at the same MySQL prompt the following:

```
source create.sql
```

If this fails, make sure you have started the MySQL prompt from the 'Data' subdirectory (this is where the database files for this example are stored).

2. Configuring the Database Connection

Edit the Java source code file by filling out JDBC information about your database, such as:

- (a) Driver Name (e.g. `com.jdbc.mysql.Driver`)

- (b) JDBC Url (e.g. `jdbc:mysql://localhost/test`)
- (c) Username and Password

If you are unsure whether you have a JDBC Driver for your Database, skip this step and try running the example with the predefined values. If that fails, you will need to probably download the latest driver.

3. Running the example

Run the ‘compile’ script (`compile.sh` for Linux, `compile.bat` for Windows) to compile the Java source code, and then the ‘run’ script to see the results.

Uninstalling the Source Data

To delete all the tables used in this example, open the MySQL prompt from the ‘Data’ subdirectory, select the database where you created the tables, and run the following:

```
source delete.sql
```

The below section provides examples for the exotic options module.

7.3 Futures Custom Examples Examples

7.3.1 Interest

Problem 1: Consider an interest rate quoted as 10% per annum with semiannual compounding. We calculate the equivalent rate with continuous compounding.

Solution

- Method used:

```
interestCompoundToContinuous(double,double)
```

- The parameters are:
 - compound interest - 0.1
 - the number of times per annum the interest is compounded - 2
- The source code:

```
double result = interestCompoundToContinuous(0.1,2);  
System.out.println(result);
```

- The result: 9.758
- Conclusion: The equivalent rate with continuous compounding is 9.758% per annum

7.3.2 FuturesEvaluation

Problem 2: Consider a four-month futures contract on a zero coupon bond that will mature one year from today. The current price of the bond is \$930. We assume that the four-month risk-free rate of interest (continuously compounded) is 6% per annum and calculate the futures price.

Solution

- Method used:

```
futuresPriceNoIncome(double,double,double)
```

- The parameters are:
 - spot price - 930
 - risk free rate - 0.06
 - time to maturity - $\frac{4}{12}$

- The source code:

```
double result = futuresPriceNoIncome(930,0.06,4/12);  
System.out.println(result);
```

- The result: 948.79
- Conclusion: \$948.79 will be the delivery price in a contract negotiated today.

Problem 3: Consider a six-month futures contract on an investment asset that is expected to provide a continuous dividend yield of 4% per annum. The risk free rate of interest (with continuous compounding) is 10% per annum. The asset's price is \$25. We calculate the futures price.

Solution

- Method used:

```
futuresPriceWithDividend(double,double,double,double)
```

- The parameters are:
 - spot price - 25
 - yield - 0.04
 - risk free rate - 0.1
 - time to maturity - 0.5

- The source code:

```
double result = futuresPriceWithDividend(25,0.04,0.1,0.5);
System.out.println(result);
```

- The result: 25.76
- Conclusion: The futures price is \$25.76.

Problem 4: Consider a four-month futures contract on the S&P 500. Suppose that the stock underlying the index provide a dividend yield of 3% per annum, that the current value of the index is 900, and that the continuously compounded risk-free interest rate is 8% per annum. We calculate the futures price.

Solution

- Method used:

```
futuresPriceOnIndex(double,double,double,double)
```

- The parameters are:
 - index value - 900
 - yield - 0.03
 - risk free rate - 0.08
 - time to maturity - 0.25
- The source code:

```
double result = futuresPriceOnIndex(900,0.03,0.08,0.25);
System.out.println(result);
```

- The result: 911.32
- Conclusions: The futures price is \$911.32.

7.3.3 Forwards

Problem 5: Consider a six month long forward contract on a non dividend paying stock. The risk free rate of interest (with continuous compounding) is 10% per annum, the stock price is \$25, and the delivery price is \$24. We calculate the long forward's price.

Solution

- Method used:

```
longForward(double,double,double,double)
```

- The parameters are:
 - spot price - 25
 - delivery price - 24
 - risk free rate - 0.1
 - time to maturity - 0.5

- The source code:

```
double result = longForward(25,24,0.1,0.5);  
System.out.println(result);
```

- The result: 2.17
- Conclusion: The long forward's value is \$2.17.

7.3.4 FuturesHedging

Problem 6: A company wishes to hedge a portfolio worth \$2,100,000 over the next three month using an S&P 500 index futures contract with four months to maturity. The current level of the S&P 500 is 900 and the β of the portfolio with respect the the S&P index is 1.5. The value of the asset underlying the futures contract is \$225,000. We calculate the number of S&P500 futures contracts needed to short in order to hedge the portfolio.

Solution

- Method used:

```
betaHedge(double,double,double)
```

- The parameters are:
 - portfolio - 2.100.000
 - index contract size - 225.000
 - beta - 1.5

- The source code:

```
double result = betaHedge(2100000,225000,1.5);  
System.out.println(result);
```


- The result: 14.
- Conclusion: The correct number of futures contracts to short is 14.

7.3.5 FuturesOnCommodities

Problem 7: Consider a one-year futures contract on gold. Suppose that it costs \$1.865 per ounce per year to store gold. Assume that the spot price is \$450 and the risk-free rate is 7% per annum for all maturities. We calculate the price of this futures contract of gold.

Solution

- Method used:

```
investmentCommodity(double,double,double,double)
```

- The parameters are:
 - commodity price - 450
 - storage cost - 1.865
 - risk free rate - 0.07
 - time to maturity - 1
- The source code:

```
double result = investmentCommodity(450,1.865,0.07,1);  
System.out.println(result);
```

- The result: 484.63
- Conclusion: The futures price is \$484.63.

7.4 Exotic Options Custom Examples

This section provides examples of using option pricing methods (in particular Monte-Carlo and Finite Differencing) implemented within the Exotic Options module of the WebCab Options and Futures J2EE Application. For the below listed examples we built working applications so that you may see how the component can be used and then how it performs.

7.4.1 About the examples

There are seven examples included for the Exotic Options module. We designed them with the intention of covering a broad range of possible user scenarios.

7.4.2 Testing procedure

In all our examples of options on one asset the following financial parameters were used:

- interest rate = 0.1
- no dividend (dividend yield = 0)

Also note that all the results are displayed with five digit precision.

Monte Carlo results will differ with each subsequent method call using identical parameters, because the Monte Carlo algorithm contains a fundamental random selection element. Therefore a comparison between MonteCarlo and finite differencing cannot be relevant unless seen in a statistical manner. You should run our examples many times before making any conclusions regarding the differences between Monte Carlo and finite differencing.

In some of the examples we use a formula to estimate the magnitude of the difference in relation with the magnitude of the results. That is, we calculate:

$$\text{relative_difference} = \frac{|\text{fd_result} - \text{mc_result}|}{|\text{fd_result}| + |\text{mc_result}|} \quad (7.4.1)$$

7.4.3 Example 1 - European vanilla options

Motivation and Problem

This example compares finite differencing with Monte Carlo for very simple option types with known boundaries. We use the *SimplePayoff* and *SimpleBoundaries* classes to specify our option type. The contract stipulates that we have an European call option with strike price equal to 50 and that it will be exercised after one year.

Suppose that the current asset price is 40 and we wish to find out the present option value.

The Parameters

In this instance the following parameters should be used within the finite differencing algorithm:

- algorithm = Cranck-Nicholson
- time steps = 1000
- maximum asset price = 150
- asset price steps = 150

Results

According to the finite difference procedure the present option price is 4.44724.

For Monte Carlo, the number of random walks is set to 1000.

Almost 90% of the tests runs with this example show a relative difference between Monte Carlo and finite differencing less than 5%.

Source Files

The source file for this example is *EuropeanVanilla/EuropeanVanilla.java* and it can be found in the */Client/ExoticOptions* directory within the Applications package.

7.4.4 Example 2 - Supplying your own payoff function and boundaries

Motivation and Problem

This example shows you how to supply your own payoff function and boundaries. The payoff function used here is actually the square root of the payoff from a Vanilla put option.

We compared the results obtained using the generic *SecondOrderBoundaries* class with the results obtained when we have set the boundaries by implementing *Dirichlet* abstract class.

The Parameters

The algorithm parameters are exactly the same as those used in the previous example.

Results

The results are:

- generic boundaries - 2.3347
- user designed boundaries - 2.34563

The Monte Carlo method returns similar results.

Source Files

The source files for this example is *SupplyingPayoffFunction/SupplyingPayoffFunction.java*. It can be found in the */Client/ExoticOptions* directory within the Applications package.

7.4.5 Example 3 - Supplying your own payoff function for multi-asset boundaries

The Problem This example is similar to the previous one, the only difference being that now we have a two-asset option to be priced.

The Parameters

The financial parameters are:

- interest rate = 0.1
- volatility of first asset = 0.4
- volatility of second asset = 0.3
- no dividend (dividend yield = 0)
- correlation matrix = $\begin{pmatrix} 1 & 0.15 \\ 0.15 & 1 \end{pmatrix}$

As there is more than one asset, we do not have the possibility of choosing the algorithm type. The only algorithm which can be used is the explicit algorithm.

The (explicit) algorithm parameters are:

- time steps = 200
- maximum asset price = 150
- asset price steps = 50

The strike price for the first asset is 50 and for the second asset it is 40.

Results

The results are:

- generic boundaries - 2.19603
- user designed boundaries - 2.19578

Remark If you are using the J2EE platform and local delivery then you can see that the finite differencing algorithm is slower in combination with the generic boundaries. The reason for this is that the computations required by the client *MyBoundaries* class are fewer than those performed by *SecondOrderBoundaries*. Quite the opposite happens when using remote delivery. In this case, with generic boundaries, the algorithm finishes much faster. Indeed, the server using client boundaries must now call the *getValueAt* method remotely while all processing for general boundaries still takes place on the server. In this instance you can see the huge speed difference which can occur between the two delivery methods. Here in this example there are 40000 calls to the boundary evaluation method which needs 40000 network connections and greatly infringes performance.

Source Files

The source files for this example is *MultiAssetOptions/MultiAssetOptions.java* and it can be found in the */Client/ExoticOptions* directory within the Applications package.

7.4.6 Example 4 - Thorough comparison between finite differencing and Monte Carlo

Motivation and Problem

In this example we call the Cranck-Nicholson finite differencing algorithm and Monte Carlo successively for 6 types of options each having 10 different maturity dates. This means that we have a total of 60 option contract to value. For each contract the absolute and the relative difference between the results are also displayed.

Remark There exist some results that are evidently wrong (for example the negative value which appears when evaluating Lookback strike put option with 10 years to maturity). The most probable cause of these and of other very large differences is simply that the boundaries are wrong (the “general” boundaries we use here are based on two assumptions that are not necessarily applicable to all option types). In such cases you should construct your own boundaries. For further discussion of this issue please see [boundary conditions](#) section of the Exotic Options Programmers Guide. Also see other possible [error sources](#) in Mathematical Documentation.

Source Files

The source file for this example is *ComparisonFD_MC/ComparisonFD_MC.java* and it can be found in the */C*

7.4.7 Example 5 - American options

Motivation and Examples

This example compares the three finite differencing algorithms: explicit, implicit and Cranck-Nicholson. American options cannot be priced with Monte Carlo, so no such results are displayed. When evaluated with an implicit algorithm (fully implicit or Cranck-Nicholson) the tolerance parameter must be set. In our example the tolerance was set to 0.001. Generally it is not worth trying to set this to a lower value in the hope that precision will increase.

The Parameters

Algorithm parameters are:

- *time steps* = 2000
- *maximum asset price* = 150

- *asset price steps* = 150

The current price was considered to be 75, the strike price 50 and the time to maturity of one year.

Results

The results are:

- *explicit algorithm* - 0
- *fully implicit algorithm* - 30.07861
- *Cranck Nicholson algorithm* - 30.51493

The explicit algorithm returns 0, because it is unstable in the current configuration of algorithm parameters. If you increase the number of time-steps to, say 10000, you will see that it will return a correct result. The same thing happens if you decrease the asset price step to 50 (for example).

Remark Zero is not the only result possible when the algorithm is unstable. Often very large results are returned (for example $1E100$).

Source Files

The source file for this example is *AmericanOptions/AmericanOptions.java* and it can be found in the */Client/ExoticOptions* directory within the components package.

7.4.8 Example 6 - American multi-asset options

Motivation and Problem

This example shows you how to evaluate American multi-asset options. For this purpose we use two standard option strategies: the spread and the strangle.

The Parameters

The financial parameters are the same as in the [European multi-asset example](#).

The algorithm parameters are:

- *time steps* = 1000
- *maximum asset price* = 100
- *asset price steps* = 50

The contract specifies that the strike price for the first asset is 50 and for the second asset 40. The expiry time is one year and the current asset prices are 40 and 90 respectively.

Results

The results are:

- American bull spread - 4.60251
- American bear spread - 10
- American strangle - 12.5145
- American straddle strategy - 10.09196

Source Files

The source file for this example is *AmericanMultiAssetOptions/AmericanMultiAssetOptions.java* and it can be found in the */Client/ExoticOptions* directory within the Applications package.

7.4.9 Example 7 - Example of Bermudan options

Motivation and Problem

This is an example of a simple put option with expiry time of one year and early exercise permitted only during an interval of six month beginning with the third month of the year. All these restrictions are set in the payoff function, which is, in this case, time dependent. That means we have a Bermudan-type option.

The Parameters

The current price is 40 and the strike price is 50. Cranck-Nicholson algorithm is used with generic boundaries.

Results

The result returned are 10.91813.

Source Files

The source files for this example is *BermudanOptions/BermudanOptions.java*. It can be found in the */Client/ExoticOptions* directory within the Applications package.

7.4.10 Example 8 - Example of Binary options

Motivation and Problem

This is an example of a binary option, which pays off 1 currency unit if the price after one year is above the strike price.

Results

The result of the Crank-Nicholson algorithm is 0.28462.

Source Files

The source files for this example is *BinaryOptions/BinaryOptions.java*. It can be found in the */Client/ExoticOptions* directory within the Application package.

7.4.11 Example 9 - Example of Asian options

Motivation and Problem

This example shows you how to use the pricing methods for strongly path dependent options. Specifically a Asian option with daily arithmetic averaging is used. It is a rate (price) call option. This means that the payoff is $\max(\text{average} - \text{strike}, 0)$. The following parameters were used:

- interest rate = 0.1
- volatility = 0.3
- time to maturity = 1
- current price = 40
- strike price = 50
- dividend = 0
- price steps = *averagesteps* = 150
- maximum price = *maximumaverage* = 150
- time steps for finite differencing = $5 * 260 = 1300$
- sampling period = $5(\text{dailyupdatingoftheaverage})$
- time steps *for MonteCarlo* = 260
- random walks = 9000

Results

The result of the Cranck-Nicholson algorithm is 0.74908.

Source Files

The source files for this example is *AsianOptions/AsianOptions.java*. It can be found in the */Client/ExoticOptions* directory within the Application package.

7.4.12 Example 10 - Example of Lookback options

Motivation and Problem

This example is very similar to the previous one, except that we are now pricing lookback options instead of Asian options (which are also strongly path dependent). The difference is that the path dependent variable is now the maximum value of the price registered from the beginning of the contract. The payoff is $\max(\max - \text{strike}, 0)$. The same parameters as in the previous example were used.

Results

The result of the Cranck-Nicholson algorithm is 4.94097.

Source Files

The source files for this example is *LookbackOptions/LookbackOptions.java*. It can be found in the */Client/LookbackOptions* directory within the Application package.

7.4.13 Example 11 - Example of Monte Carlo with error control

Motivation and Problem

This example shows you how to use error control for Monte Carlo. It is identical with the first example (European vanilla options), except that now we require that the relative error of Monte Carlo is less than 5%, with 95% certainty. So the confidence level is 0.95 and `max_relative_error` is 0.05.

Results

If you run the example multiple times, you can notice that in the large majority of cases the relative error is below 5%. On the average, only 1 in 20 runs should have a bigger error.

Source Files

The source files for this example is *ErrorControl/ErrorControl.java*. It can be found in the */Client/LookbackOptions* directory within the Application package.

Chapter 8

Guide to WebCab Components

8.1 The Company

WebCab is a privately owned British company that has built business solutions since its inception in 1999. We continue to refine and develop our Mathematical and Financial Framework which we have implemented within the J2SE, J2EE and .NET platforms.

8.2 Presentation of Products

We aim to provide good quality, useful information to help you decide which J2SE Component best suits your development needs. WebCab is committed to honesty and realism when presenting our products. In order to achieve this a detailed, clear and factual style for presentation is adopted within our documentation and marketing material.

8.3 Supported Clients, IDEs, Containers and DBMSs

We support all major development, server and client side technologies. Each product contains detailed examples and advice concerning the integration of our components within existing infrastructure. Our documentation provides the information that the developer needs to get their applications up and running as quickly and as easily as possible.

We detail exactly how the developer can use our J2SE Components with the following technologies:

- Client containers - Internet Explorer, Netscape Navigator, Mozilla, Opera
- IDEs - JBuilder, BEA Studio, Sun ONE (formerly Forte For Java), Eclipse, Oracle JDeveloper
- Client Side Technologies - Application Clients, Frames, Applets
- DBMS - Oracle, IBM DB2, SQL Server, Sybase, MySQL

- Platforms - Windows XP/2000/NT, Linux, Solaris, IBM AIX, HP-UX

For these technologies we include all the installation scripts and template examples which will help the developer quickly and easily assemble their multi-tier enterprise application.

8.4 Transparent Functionality

All technical and business intelligence incorporated within our products is described within the associated documentation. This allows the developer to see the nature of the methodology implemented within our proprietary algorithms.

8.5 Code Conventions

WebCab adheres to the ‘Code Conventions for the Java Programming Language’ as specified on April 20, 1999 by Sun Microsystems Inc. These conventions cover filenames, file organization, indentation, comments, declarations, statements, white space, naming conventions and programming practices. Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

8.6 Company Culture and Activity

WebCab Components is focused solely on the production of high quality software modules within our Mathematical and Financial Framework. The WebCab team contains a wide range of expertise and experience within the academic, investment banking and software development worlds.

Within the company there exists significant internal competitive pressures with constant peer review and evaluation, resulting in higher quality products, which adhere to high professional standards and offer extended functionality.

8.7 Product Life cycle

We continuously add value to our products by evolving them according to new J2SE specifications, customer feedback and market demands. We give particular emphasis to incorporating our clients suggested modifications and additions into our product development cycle.

8.8 Support, Warranty and Upgrades

WebCab warrants that each component will perform substantially in accordance with the accompanying written material. We provide with all our products without charge sixty

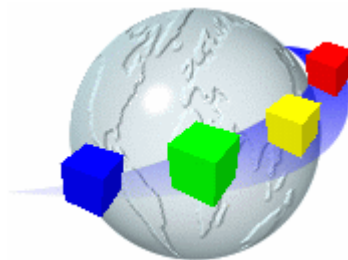
(60) days product support services including fixing bugs, compatibility issues and other technical support issues.

All maintenance updates (including service packs) will be distributed free of additional license cost. New version releases of this product will be offered to present licensee holders at a greatly reduced rate.

Dr Ben Fairfax

Founder and CEO

WebCab Components - From Developer, To Developer



Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.