
# Topics in Empirical Finance with R and Rmetrics

Patrick Hénaff

Rmetrics Association & Finance Online

R/Rmetrics eBook Series

R/Rmetrics eBooks is a series of electronic books and user guides aimed at students and practitioner who use R/Rmetrics to analyze financial markets.

*A Discussion of Time Series Objects for R in Finance* (2009)
Diethelm Würtz, Yohan Chalabi, Andrew Ellis

*Portfolio Optimization with R/Rmetrics* (2010),
Diethelm Würtz, William Chen, Yohan Chalabi, Andrew Ellis

*Basic R for Finance* (2010),
Diethelm Würtz, Yohan Chalabi, Longhow Lam, Andrew Ellis
Early Bird Edition

*Financial Market Data for R/Rmetrics* (2010)
Diethelm Würtz, Andrew Ellis, Yohan Chalabi
Early Bird Edition

*Indian Financial Market Data for R/Rmetrics* (2010)
Diethelm Würtz, Mahendra Mehta, Andrew Ellis, Yohan Chalabi

# Topics in Empirical Finance with R and Rmetrics

Patrick Hénaff

Rmetrics Association & Finance Online

*Series Editors:*
PD Dr. Diethelm Würtz
Institute of Theoretical Physics and
Curriculum for Computational Science
Swiss Federal Institute of Technology
Hönggerberg, HIT K 32.2
8093 Zurich

*Contact Address:*
Rmetrics Association
Weinbergstrasse 41
8006 Zurich
info@rmetrics.org

*Authors:*
Yohan Chalabi, ETH Zurich
Mahendra Mehta, NeuralTechSoft Mumbai
Andrew Ellis, Finance Online GmbH Zurich
Diethelm Würtz, ETH Zurich

Dr. Martin Hanf
Finance Online GmbH
Weinbergstrasse 41
8006 Zurich

*Publisher:*
Finance Online GmbH
Swiss Information Technologies
Weinbergstrasse 41
8006 Zurich

ISBN:
eISBN:
DOI:

# Dedication

*To E.M.O. for their love of science,*
*and to ABH for her love of books.*

# PREFACE

*Une totale soumission aux données de l'expérience*
*est la règle d'or qui domine toute discipline,*
*toute activité valable.*
Maurice Allais
lycée Lakanal
February 3, 2001

These notes are about empirical finance, and focuss on the pricing and risk management of financial assets: bonds, futures contracts, and other derivative securities.
Following Maurice Allais's advise, the emphasis is empirical. We present models, discuss their implementation, and verify their relevance by testing them of real data. Throughout the text, we emphasize:

- an incremental approach to model building, starting from simple models, and building upon that foundation to construct more complex models, as needed,
- a data-driven approach: we will implement all the models that are presented, using the R statistical package and the Rmetrics libraries,
- the systematic use of simulation as a way of validating modeling decisions or measuring various modeling biases.

This text would not be possible without the R statistical program and without the numerous Rmetrics packages. We extend our deep appreciation to the R community and to Diethelm Wuertz and the Rmetrics team.
Finally, these notes grew out of courses in computational finances given at Télécom-Bretagne and EURIA (Université de Bretagne Occidentale), both situated in Brest, France, as well as at IAE-Paris (Université Paris I Panthéon-Sorbonne). The students there deserve a special thank you for enduring early drafts of the manuscript.

Patrick Hénaff
St Jean Trolimon, Winter 2012

# Contents


Dedication iii

Preface v

Contents vii

List of Figures x

List of Listings xii

List of Tables xiv

**I The Experimental Environment 1**

1 Data Sources 5
- 1.1 *Commodities Data* . . . 5
- 1.2 *Libor Rates* . . . 7
- 1.3 *Equities Data* . . . 8

2 Basic Components 9
- 2.1 *The `fInstrument` Class* . . . 9
- 2.2 *The DataProvider Class* . . . 13

3 The Simulation Framework 15
- 3.1 *Scenario Simulator* . . . 15
- 3.2 *A Delta Hedging Experiment* . . . 16

**II Basic Fixed Income Instruments 21**

4 Elementary Fixed Income Calculations 23
- 4.1 *Bond Pricing* . . . 23
- 4.2 *Basic Measures of Interest Rate Risk* . . . 25
- 4.3 *Bond Clean and Dirty Price* . . . 31

5 THE TERM STRUCTURE OF INTEREST RATES 33
5.1 *Term Structure Shapes* . . . . . 33
5.2 *Building a Zero-Coupon Bond Curve* . . . . . 34
5.3 *Bootstrapping a Zero-Coupon Libor Curve* . . . . . 43

6 FIXED INCOME RISK MANAGEMENT 47
6.1 *Immunization* . . . . . 47
6.2 *Dedication* . . . . . 51

**III Discreet Models for Derivatives 57**

7 RISK-NEUTRAL PRICING IN A BINOMIAL FRAMEWORK 59
7.1 *Introduction* . . . . . 59
7.2 *Pricing With Arrow-Debreu Securities* . . . . . 61
7.3 *Multi-Period Binomial Model* . . . . . 62
7.4 *American Exercise* . . . . . 63
7.5 *Calibration of the Binomial Model* . . . . . 64
7.6 *Stability of the Binomial Model* . . . . . 68
7.7 *Trinomial Models* . . . . . 73

**IV Derivatives in the Black-Scholes Framework 75**

8 PRICE AND GREEKS IN THE BLACK-SCHOLES MODEL 77
8.1 *Solving the Pricing Equation* . . . . . 77
8.2 *An Option Dashboard* . . . . . 79

9 DYNAMIC HEDGING IN THE BLACK-SCHOLES FRAMEWORK 87
9.1 *Self-Financing Replicating Portfolio* . . . . . 87
9.2 *Hedging Error due to Discrete Hedging* . . . . . 94
9.3 *Hedging Error due to Transaction Cost* . . . . . 100
9.4 *Hedging Error due to Unknown Volatility* . . . . . 103
9.5 *Conclusion* . . . . . 105

10 VANNA-VOLGA PRICING AND HEDGING 107
10.1 *Principle* . . . . . 107
10.2 *Implementation* . . . . . 110
10.3 *Illustrations* . . . . . 111

**V Stylized Facts on Asset Returns 119**

11 STYLIZED FACTS OF FINANCIAL DATA 121
11.1 *Distribution of Returns* . . . . . 122

11.2 *Fitting a Density to Historical Returns* . . . . . 125
11.3 *Autocorrelation of Returns* . . . . . 127

**VI The Volatility Surface** **131**

12 THE IMPLIED VOLATILITY 133
12.1 *Market Data* . . . . . 133
12.2 *Calculation of implied volatility* . . . . . 134

13 BUILDING AN EQUITY VOLATILTY SURFACE 141
13.1 *The Data* . . . . . 141
13.2 *Estimation of the Volatility Surface* . . . . . 143

14 THE IMPLIED DISTRIBUTION OF ASSET PRICE 149
14.1 *The Breeden-Litzenberger Formula* . . . . . 149
14.2 *Analytical expression for the density of* $S_T$ . . . . . 151

15 IMPLIED TREES 165
15.1 *The Derman-Kani Implied Tree* . . . . . 165

BIBLIOGRAPHY 173

INDEX 175

ABOUT THE AUTHOR 177

# List of Figures


2.1 Call Price as a function of spot value . . . . . 14

3.1 Simulation of log-normal price process, annual volatility: 30%. 16
3.2 Distribution of delta-hedging error . . . . . 18

4.1 Price vs. yield as a function of coupon rate . . . . . 25
4.2 Price vs. yield as a function of maturity . . . . . 26
4.3 Bond duration by maturity . . . . . 30
4.4 Clean and dirty prices of par and premium bonds . . . . . 32

5.1 ECB Yield Curves: 2006-2009 . . . . . 35
5.2 US Treasury Yield Curves: 1982-2009 . . . . . 36
5.3 YTM, Par and zero-coupon curves . . . . . 38
5.4 Fitted US Treasury zero-coupon curve . . . . . 40
5.5 Fitted US Treasury forward curves . . . . . 41
5.6 Components of Fitted Treasury zero-coupon spot curves, with Nelson-Siegel model . . . . . 42
5.7 Principal components analysis of US zero rates . . . . . 43
5.8 Factor loadings, US zero rates . . . . . 44
5.9 USD zero-coupon LIBOR curve, bootstrapped from deposit and swap rates . . . . . 46

6.1 Scenario analysis of immunized liability . . . . . 50

7.1 A two-period binomial tree . . . . . 62
7.2 2-steps binomial tree . . . . . 63
7.3 2-step binomial tree . . . . . 65
7.4 Convergence of CRR to Black-Scholes . . . . . 70
7.5 Convergence of European Call in Tian's Binomial Model . . . . . 70
7.6 Convergence of CRR with Drift to Black-Scholes . . . . . 72
7.7 Accelerated Convergence of the Binomial Model . . . . . 73

8.1 Price and greeks on a European call . . . . . 81
8.2 European Binary Call Strike=100 . . . . . 83

8.3 European Barrier Call (UAO) Strike=100 . . . . . 85
8.4 European Asian Call Strike=100 . . . . . 86

9.1 Simulation of a delta-hedging strategy . . . . . 94
9.2 Delta hedging error vs. hedging frequency . . . . . 97
9.3 Vanilla vs. Binary Options . . . . . 99
9.4 Delta Hedging Errors: Vanilla and Binary Calls . . . . . 101
9.5 Delta hedging error vs. transaction cost . . . . . 103

10.1 Vanna-Volga Volatility Interpolation . . . . . 113
10.2 Pricing a binary call with Vanna-Volga method . . . . . 117

11.1 Daily time series of WTI Futures, Dec 2009 expiry . . . . . 122
11.2 Daily returns series, WTI Futures Dec 2009 . . . . . 123
11.3 Histogram of daily return, WTI Futures . . . . . 124
11.4 QQ plot of daily returns . . . . . 125
11.5 Histogram of daily returns and fitted Johnson SU distribution . 127
11.6 Johnson vs. Normal distribution fitted to daily WTI Futures returns . . . . . 128

13.1 Imnplied volatility surface, SPX index options, 24-Jan-2011 . . . 147

14.1 Quadratic model fitted to implied volatility . . . . . 156
14.2 Implied density from smile . . . . . 157
14.3 Effect of volatility smile on the price of a digital option . . . . . 159
14.4 Implied Volatility, WTI Nymex Options . . . . . 160
14.5 Density of WTI Futures at expiry implied by option smile . . . . 162
14.6 Implied density of WTI Futures price, fitted to a Johnson distribution . . . . . 163

15.1 One step of a trinomial tree . . . . . 166
15.2 Constant Volatility Trinomial Tree . . . . . 168
15.3 Transition probabilities in Derman-Kani implied tree . . . . . 170
15.4 BlackScholes and local volatilities in Derman-Kani tree . . . . . 171

# LIST OF LISTINGS

12.1 Option settlement prices. source: NYMEX . . . . . 133

# List of Tables


3.1 Delta-hedging of a Europen call, strike: 100 . . . . . 19

4.1 Bond portfolio . . . . . 27

5.1 Calculation of a Treasury Zero-Coupon Curve . . . . . 36
5.2 USD Libor and deposit rates, Nov 10, 2009 . . . . . 44

6.1 . . . . . 48
6.2 Liability cash flow stream . . . . . 51
6.3 Available bonds for dedication . . . . . 51
6.4 Dedicated Portfolio . . . . . 53
6.5 Bond dedication . . . . . 54
6.6 Bond dedication . . . . . 55

7.1 Future states of a one period binomial model . . . . . 60

9.1 Delta-hedging of a Europen call, strike: 100 . . . . . 93
9.2 Delta-hedging of a Europen call, strike: 100 . . . . . 93
9.3 Moments of hedging error: Vanilla vs. Binary . . . . . 101
9.4 Fitness of the Black-Scholes Model . . . . . 105

10.1 Volatility data . . . . . 111

11.1 Sample moments of return: WTI Futures . . . . . 124
11.2 Fitted Johnson SU parameters . . . . . 126
11.3 Sample and fitted moments with Johnson density . . . . . 126

12.1 Implied volatility timing test . . . . . 136
12.2 Implied volatility timing test for ITM options . . . . . 138
12.3 Implied volatility timing test with Jaeckel's parametrization . . 140

13.1 SP 500 Index option quotes . . . . . 142

14.1 Option prices (Shimko's 1993 dataset) . . . . . 153

# Part I

# The Experimental Environment

This first part is dedicated to the description of the experimental environment used in this book.
As mentioned earlier, our approach is data-driven and empirical. It is thus appropriate to start this text with a description of the sources of data that we will use, and how to fetch publicly available financial data from the Internet.
In the following chapters, we will make repeated use of the Rmetrics pricing libraries, and often compare different models applied to the same data. To facilitate this comparison, the Rmetrics pricing libraries have been wrapped in a object-oriented framework, which hides most of the implementation details, and allows us to focus on the key features of the financial instruments and models. The framework uses the S4 object model (Genolini, 2008) of the R language, and is described in chapter 2.
Simulation is our tool of choice to explore the features of pricing models and test their robustness. To that end, we have developped a framework that facilitates the definition and testing of simple risk management strategies. The core of this simulation framework is the `DataProvider` class, which is presented in Chapter 3.
In addition the packages found on CRAN, data sets and code used in the text have been gathered into three packages:

**`empfin`** contains all the data sets and several utility functions for simple bond pricing and the manipulation of dates,

**`fInstrument`** provides the `fInstrument` class, that wraps the Rmetrics pricing library and the `DataProvider` class, that acts as a container for market data,

**`DynamicSimulation`** contains tools needed to perform dynamic simulations, such as scenario generators and delta hedging simulators.

# Chapter 1

# Data Sources

```
> library(RCurl)
> library(fImport)
> library(timeSeries)
> library(empfin)
> data(LiborRates)
```

This first chapter describes the data sources and the processing that has been performed to create the data sets found in the `empfin` package. In the following chapters, we will use data from three asset classes: commodities, equities and fixed income.

## 1.1 Commodities Data

The CME group (www.cmegroup.com) provides a large quantity of data on the products listed on its exchanges. Commodity prices are particularly interesting in empirical finance because they exhibit high volatility, seasonality, and many other specific features.
The following function downloads settlement data on commodity options and futures traded on the NYMEX.

```
getNymexData <- function(type) {

url <- 'ftp://ftp.cmegroup.com/pub/settle/'
filename <- paste('nymex_', type, '.csv', sep='')
fn <- paste(url, filename, sep='')

curl <- getCurlHandle(ftp.use.epsv=FALSE)
x <-  getURL(fn, curl=curl)

csv <- strsplit(x, '\n')[[1]]
```

```
con <- textConnection(csv)
tmp <- data.frame(read.csv(con, header=TRUE))
close(con)

if(type == 'future') {
  goodColumns <- c('PRODUCT.SYMBOL',
                   'CONTRACT.MONTH',
                   'CONTRACT.YEAR',
                   'CONTRACT',
                   'PRODUCT.DESCRIPTION',
                   'SETTLE',
                   'EST..VOL',
                   'TRADEDATE')

  goodNames <- c('Symbol', 'Month', 'Year', 'Contract',
                   'Description', 'Settle', 'Volume', 'TradeDate')

}

if(type == 'option') {
  goodColumns <- c('PRODUCT.SYMBOL',
                   'CONTRACT.MONTH',
                   'CONTRACT.YEAR',
                   'PUT.CALL',
                   'STRIKE',
                   'SETTLE',
                   'EST..VOL',
                   'TRADEDATE')

  goodNames <- c('Symbol', 'Month', 'Year',
                   'CP', 'Strike',
                   'Settle', 'Volume', 'TradeDate')
  }
tmp <- tmp[,goodColumns]
colnames(tmp) <- goodNames
tmp
}
```

As an illustration, the following script downloads the NYMEX futures data and extracts the Crude Oil (WTI) futures settlement prices. Running the code on October 11, 2012, yields the following result:

```
> nymex.futures <- getNymexData('future')
> tmp <- nymex.futures[nymex.futures$Symbol == 'CL',]
> head(tmp[,c('Month', 'Year', 'Contract', 'Settle')])

     Month Year Contract Settle
1998    11 2012    CLX12  91.25
1999    12 2012    CLZ12  91.64
2000     1 2013    CLF13  92.11
2001     2 2013    CLG13  92.58
2002     3 2013    CLH13  93.00
2003     4 2013    CLJ13  93.30
```

The U.S. Energy Information Administration (www.eia.gov) publishes historical data on spot and future prices. The data set `CL-Historical` in package `empfin` was obtained from that source.

## 1.2 LIBOR RATES

The `empfin` package also contains a data set of USD libor and swap rates. The data comes from the US Federal Reserve archive (www.federalreserve.gov). The time series can be downloaded from a web browser, but can also be downloaded programmatically. The site provides directions on how to construct the URL corresponding to each particular data set.
In the example below, the URL is specific to the H15 table, with all available deposit and swap rates included.

```
get.frb.url <- function(dtStart, dtEnd) {

    # Federal Reserve Board URL
    # Construct this URL at 'http://www.federalreserve.gov/datadownload

    url.1 <- 'http://www.federalreserve.gov/datadownload/Output.aspx?rel=
         H15&series=8f47c9df920bbb475f402efa44f35c29&lastObs=&from='
    url.2 <- '&filetype=csv&label=include&layout=seriescolumn'
    url <- paste(url.1, as.character(dtStart, format='%m/%d/%Y'), '&to=',
                 as.character(dtEnd, format='%m/%d/%Y'), url.2, sep='')
    url
    }
```

The data is downloaded from the FRB web site and converted into a `time-series` object, with more explicit names. The resulting time series has daily observations and 11 columns, with deposit rates of maturities 1, 3, and 6 months, and swap rates with maturities ranging from 1 to 30 years.

```
columns.dic = hash(keys=c("RIFLDIY01_N.B",
                          "RIFLDIY02_N.B",
                          "RIFLDIY03_N.B",
                          "RIFLDIY04_N.B",
                          "RIFLDIY05_N.B",
                          "RIFLDIY07_N.B",
                          "RIFLDIY10_N.B",
                          "RIFLDIY30_N.B",
                          "RILSPDEPM01_N.B",
                          "RILSPDEPM03_N.B",
                          "RILSPDEPM06_N.B"),
                   values = c('Swap1Y', 'Swap2Y',
                     'Swap3Y', 'Swap4Y', 'Swap5Y',
                     'Swap7Y', 'Swap10Y', 'Swap30Y',
                     'Libor1M', 'Libor3M', 'Libor6M'))


getLiborRates <- function(dtStart, dtEnd) {

  # non-available data is marked 'ND' or 'NC' in the data set
```

```
  good.row <- function(x) length(grep('NC|ND', x)) == 0

  url <- get.frb.url(dtStart, dtEnd)

  curl <- getCurlHandle(ftp.use.epsv=FALSE)
  x <-  getURL(url, curl=curl)

  csv <- strsplit(x, '\n')[[1]]

  # skip 5 header rows at start of file
  con <- textConnection(csv[6:length(csv)])
  # colClasses='character' needed to avoid conversion to factors
  csv <- read.csv(con, colClasses='character', header=TRUE)
  close(con)

  ncol <- dim(csv)[2]
  indx <- apply(csv, 1, good.row)

  dt <- as.Date(csv$Time.Period[indx])
  vx <- data.frame(csv[indx,2:ncol])
  for(i in seq(ncol-1)) vx[,i] = as.numeric(vx[,i])

  colNames = sapply(names(vx), function(x) columns.dic[[x]])
  timeSeries(vx, dt, format='%Y-%m-%d', zone='GMT', FinCenter='GMT', units=
       colNames)
}
```

Executing this function on October 13, 2012 yields the following result.

```
> dtStart <- myDate('01Jan2012')
> dtEnd <- myDate('01Jun2012')
> ts = getLiborRates(dtStart, dtEnd)
> head(ts[,c('Swap1Y', 'Swap10Y', 'Swap30Y')])
GMT
           Swap1Y Swap10Y Swap30Y
2012-01-03   0.68    2.10    2.69
2012-01-04   0.68    2.10    2.71
2012-01-05   0.68    2.10    2.72
2012-01-06   0.64    2.10    2.72
2012-01-09   0.62    2.08    2.70
2012-01-10   0.61    2.11    2.74
```

The dataset `LiborRates` in package `empfin` has been constructed with this function, and provides daily rates from July 2000 to October 2010.

## 1.3 EQUITIES DATA

Time series of stock prices are readily available from free on-line services. For illustrative purpose, the `empfin` package provides one time series (`EquitySeries`) of daily prices for three stocks: IBM, Coca-Cola (KO) and Trans-Ocean (RIG), and one Exchange Traded Fund: the Nasdaq 100 Trust (QQQQ).

# CHAPTER 2

# BASIC COMPONENTS

This chapter provides a tutorial and explains the design of the object framework that has been build on top of the Rmetrics library. As mentioned earlier, this layer is meant to hide most of the implementation details, and allows us to focus on the key features of the financial instruments and models.
The reader is encouraged to work through the examples, but the sections on design and implementation can be skipped.
The object framework involves two main entities:

1. the `fInstrument` class models an abstract financial instrument, and exposes generic methods for computing the net present value (NPV) and the risk indicators (the "greeks"). With this class, one can perform calculations on portfolio of instruments, without being concerned about the implementation details specific to each type of asset. This will be illustrated below with the detailed description of a delta-hedging algorithm.

2. the `DataProvider` class is a container for market data, derived from the built-in `R environment`. It greatly simplifies the signature of pricing functions. Instead of passing as arguments all the necessary market data, we simply pass one `DataProvider` argument. The pricing methods fetch the necessary data from the `DataProvider`, as needed.

Each entity is now described and illustrated.

## 2.1 THE `fInstrument` CLASS

As mentioned, the purpose of the `fInstrument` class is to create a layer of abstraction over the large variety of pricing models found in Rmetrics.

With this class, we can express calculation algorithms in a generic manner. This is best explained by an example.

*Illustration*

Consider a portfolio made of two options, a vanilla European call and a binary (cash-or-nothing) option, both written on the same underlying asset. We would like to compute the NPV and delta of this portfolio. Let's contrast the process, first performed with the Rmetrics functions, and then with the `fInstrument` class.
Starting with the Rmetrics functions, you first compute the price and delta of the European call:

```
> cp <- "c"
> Spot <- 100
> Strike <- 100
> Ttm <- 1
> int.rate <- 0.02
> div.yield <- 0.02
> sigma <- 0.3
> p[1] <- GBSOption(TypeFlag = cp, S = Spot, X = Strike, Time = Ttm,
      r = int.rate, b = int.rate - div.yield, sigma = sigma)@price
> d[1] <- GBSGreeks(Selection = "delta", TypeFlag = cp, S = Spot,
      X = Strike, Time = Ttm, r = int.rate, b = int.rate - div.yield,
      sigma = sigma)
```

Perform the same calculation for the binary option. The delta is computed by finite difference.

```
> K <- 1
> p[2] <- CashOrNothingOption(TypeFlag = cp, S = Spot, X = Strike,
      K = K, Time = Ttm, r = int.rate, b = int.rate - div.yield,
      sigma = sigma)@price
> h <- Spot * 0.001
> dh <- CashOrNothingOption(TypeFlag = cp, S = c(Spot + h, Spot -
      h), X = Strike, K = K, Time = Ttm, r = int.rate, b = int.rate -
      div.yield, sigma = sigma)@price
> d[2] <- diff(dh)/(2 * h)
```

Finally, sum both vectors to get the portfolio NPV and delta.

```
> print(paste("Price:", round(sum(p), 2), "Delta:", round(sum(d),
      3)))
[1] "Price: 12.14 Delta: 0.536"
```

With the `fInstrument` class, the calculation steps are quite different.
You first create a vanilla instrument with the `fInstrumentFactory` function:

```
> dtExpiry <- myDate("01jan2011")
> underlying <- "IBM"
> Strike <- 100
```

```
> K <- 1
> b <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
    strike = Strike, dtExpiry = dtExpiry, underlying = underlying,
    discountRef = "USD.LIBOR", trace = F))
```

Next, use again the `fInstrumentfactory` to create the binary option:

```
> v <- fInstrumentFactory("binary", quantity = 1, params = list(cp = "c",
    strike = Strike, dtExpiry = dtExpiry, K = K, underlying = underlying,
    discountRef = "USD.LIBOR", trace = F))
```

Insert the relevant market data into a `DataProvider` (this will be explained in the next section):

```
> base.env <- DataProvider()
> dtCalc <- myDate("01jan2010")
> setData(base.env, underlying, "Price", dtCalc, 100)
> setData(base.env, underlying, "DivYield", dtCalc, div.yield)
> setData(base.env, underlying, "ATMVol", dtCalc, sigma)
> setData(base.env, "USD.LIBOR", "Yield", dtCalc, int.rate)
```

Construct a portfolio, as a list of `fInstrument` objects:

```
> portfolio = c(v, b)
```

and finally compute the price and delta of the portfolio:

```
> price <- sum(sapply(portfolio, function(a) getValue(a, "Price",
    dtCalc, base.env)))
> delta <- sum(sapply(portfolio, function(a) getValue(a, "Delta",
    dtCalc, base.env)))
> print(paste("Price:", round(price, 2), "Delta:", round(delta,
    3)))
[1] "Price: 13.31 Delta: 0.599"
```

*Design and Implementation*

`finstrument` objects are instantiated by the `fInstrumentfactory` class method, which takes as argument:

- the type of instrument being instantiated,
- the quantity or nominal amount of the instrument
- a list of parameters that define the instrument

The `fInstrumentFactory` method is simply a switch that delegates the object instantiation the the concrete subclasses of `fInstrument`. The following code fragment is extracted from the file `fInstrument.r` in package `fInstrument`:

```
  fInstrumentFactory <- function(type, quantity, params){

switch(toupper(type),
  VANILLA = Vanilla(quantity, params),
```

```
  BINARY = Binary(quantity, params),
  ASIAN = Asian(quantity, params),
  STANDARDBARRIER = StandardBarrier(quantity, params)
)
}
```

There is only one method defined on `fInstrument` objects. This method is `getValue`, and takes as argument:

- the kind of calculation being requested (price, delta)
- the calculation date,
- the data container from which the required market data will be fetched.

Again, this method simply delegates to the concrete classes the requested calculation:

```
  setMethod(f="getValue", signature=signature("fInstrument"),
   definition=function(object, selection, dtCalc, env=NULL){

   res <- NULL
   res <- switch(toupper(selection),
   PRICE = object@p(dtCalc, env),
   DELTA = object@d(dtCalc, env),
   GAMMA = object@g(dtCalc, env),
   VEGA = object@v(dtCalc, env))
   return(res*object@quantity)
})
```

As an illustration, the price calculation for vanilla options is implemented as follows in `Vanilla.r`:

```
getP <- function(dtCalc, env) {
  # get spot value
  Spot <- getData(env, Underlying, 'Price', dtCalc)
  s <- getData(env, Underlying, 'ATMVol', dtCalc)
  r <- getData(env, df, 'Yield', dtCalc)
  b <- getData(env, Underlying, 'DivYield', dtCalc)
  t <- tDiff(dtCalc, dtExpiry)
  if (trace) {
    print(paste('Calling GBSOption with Spot=', Spot, 'Strike=', Strike, 't
        =', t, 'r=', r, 'b=', b, 'sigma=', s))
    }
  GBSOption(TypeFlag=cp, S=Spot, X=Strike, Time=t, r=r, b=b, sigma=s)@price
}
```

The actual calculation being performed by the Rmetrics `GBSOption` function. The model can be easily extended to accommodate other instruments.

## 2.2 THE DATAPROVIDER CLASS

The `DataProvider` class is a container of market data, from which the pricing algorithm will fetch the necessary market information, as illustrated in the code fragment above. We first describe the representation of market data, then the algorithm for searching data in a `DataProvider`.

### *The Model for Market Data*

The model for market data is strongly inspired by Fowler (1996). To summarize, a piece of market data is modeled as an observed phenomenon on a financial instrument. Therefore, every market data item is identified by three attributes:

1. the financial instrument being observed (e.g. a stock)
2. the observed phenomenon (e.g. the implied volatility, or the price)
3. the observation date

In order to optimize storage, the data is stored in a hash table. The first two attributes are combined to created the key, and the data for all observation dates is stored as a time series, with one column for actual data, and many additional columns when performing a simulation.

### *The Search Algorithm*

The `DataProvider` inherits from the built-in `environment` class. In particular, it inherits the parent/child relationship: if a `DataProvider` has a parent, the data not found in the child environment is fetched from the parent, if possible, or from the grand-parent, and so forth.
This is useful when performing a scenario analysis where only a few variables are modified: The data held constant is stored in the parent scenario, and the modified data is stored in the child scenario which is passed as argument. This scheme is illustrated by the following example.
Let's define a vanilla option:

```
> dtExpiry <- myDate("01jan2011")
> underlying <- "IBM"
> K <- 100
> a <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
     strike = K, dtExpiry = dtExpiry, underlying = underlying,
     discountRef = "USD.LIBOR", trace = F))
```

and populate a DataProvider with the necessary market data:

```
> base.env <- DataProvider()
> dtCalc <- myDate("01jan2010")
> setData(base.env, underlying, "Price", dtCalc, 100)
```

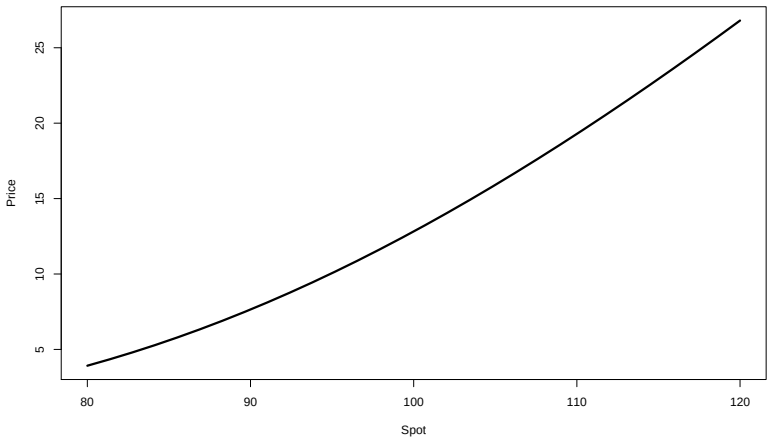



FIGURE 2.1: Call Price as a function of spot value. Strike: 100, maturity: 1 Year

```
> setData(base.env, underlying, "DivYield", dtCalc, 0.02)
> setData(base.env, underlying, "ATMVol", dtCalc, 0.3)
> setData(base.env, "USD.LIBOR", "Yield", dtCalc, 0.02)
```

The NPV of the derivative is obtained by:

```
> getValue(a, "Price", dtCalc, base.env)
[1] 12.822
```

Next, we investigate the relationship between the underlying price and the value of the derivative, by inserting a set of scenarios for the underlying asset price in a child `DataProvider`:

```
> sce <- t(as.matrix(seq(80, 120, length.out = 30)))
> sce.env <- DataProvider(parent = base.env)
> setData(sce.env, underlying, "Price", dtCalc, sce)
```

and compute the NPV of the derivative for each scenario. The relationship between the underlying price and the value of the call is illustrated in figure 2.1.

```
> p <- getValue(a, "Price", dtCalc, sce.env)
> plot(sce, p, type = "l", lwd = 3, xlab = "Spot", ylab = "Price")
```

# CHAPTER 3

# THE SIMULATION FRAMEWORK

The simulation framework has two main components:

1. A simulator, which can generate paths for the variables of interest (price of a asset, implied volatility, term structure of interest rate, etc.), according to a model that relates the variables to stochastic risk factors.

2. A framework for expressing trading strategies, and in particular dynamic hedging policies.

This is again best explained by an example, and we describe next the use of this framework for running a delta-hedging experiment. The main elements of the design are discussed afterwards.

## 3.1 SCENARIO SIMULATOR

Assume that you want to simulate 500 price paths over a period of one year, with 100 time steps. The price process will be log-normal with annual volatility of 30%, starting from an initial value of $100.

```
> dtStart <- myDate("01jan2010")
> dtEnd <- myDate("01jan2011")
> nbSteps <- 100
> nbPaths <- 500
> dtSim <- seq(dtStart, dtEnd, length.out = nbSteps + 1)
> sigma <- 0.3
> tSpot <- pathSimulator(dtSim = dtSim, nbPaths = nbPaths, innovations.gen = sobolInnovations,
      path.gen = logNormal, path.param = list(mu = 0, sigma = sigma),
      S0 = 100, antithetic = F, standardization = TRUE, trace = F)
```

The output of the path simulator is a `timeSeries`. A plot of the first few paths is displayed in Figure 3.1. The `pathSimulator` function can gen-

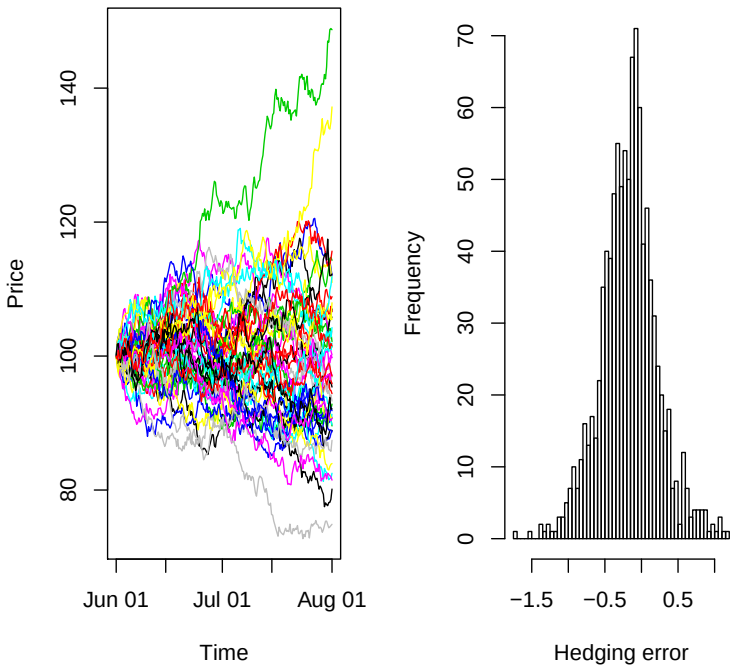



FIGURE 3.1: Simulation of log-normal price process, annual volatility: 30%.

erate simulated values acording to various statistical processes; this is documented in the vignette of the `DynamicSimulation` package.

```
> plot(tSpot[, 1:50], plot.type = "single", ylab = "Price", format = "%b %d")
```

## 3.2 A DELTA HEDGING EXPERIMENT

Having generated some scenarios for the stock price, let's now simulate the dynamic hedging of a European call option written on this stock, using the Black-Scholes pricing model, with the implied volatility and interest rate held constant.
Fist, we define the instrument to be hedged:

```
> dtExpiry <- dtEnd
> underlying <- "IBM"
> K <- 100
> a <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
```

```
    strike = K, dtExpiry = dtExpiry, underlying = underlying,
    discountRef = "USD.LIBOR", trace = FALSE))
```

Next, we define the market data that will be held constant during the simulation, and insert it in a `DataProvider`:

```
> base.env <- DataProvider()
> setData(base.env, underlying, "Price", dtStart, 100)
> setData(base.env, underlying, "DivYield", dtStart, 0.02)
> setData(base.env, underlying, "ATMVol", dtStart, sigma)
> setData(base.env, underlying, "discountRef", dtStart, "USD.LIBOR")
> setData(base.env, "USD.LIBOR", "Yield", dtStart, 0.02)
```

At this stage, we can price the asset as of the start date of the simulation:

```
> p <- getValue(a, "Price", dtStart, base.env)
```

which gives a value of $p = 12.82$.
Next, define the simulation parameters: we want to simulate a dynamic hedging policy over 500 paths, and 100 time steps per path:
We use a child `DataProvider` to store the simulated paths. Data not found in the child `DataProvider` will be searched for (and found) in the parent (`base.env`).

```
> sce.env <- DataProvider(parent = base.env)
> setData(sce.env, underlying, "Price", time(tSpot), as.matrix(tSpot))
```

We can now run the delta-hedge strategy along each path:

```
> assets = list(a)
> res <- deltaHedge(assets, sce.env, params = list(dtSim = time(tSpot),
      transaction.cost = 0), trace = F)
```

The result is a data structure that containts the residual wealth (hedging error) per scenario and time step. The distribution of hedging error at expiry is shown in Figure 3.2.

```
> hist(tail(res$wealth, 1), 50, xlab = "wealth", main = "")
```

To better illustrate the hedging policy, let's run a toy example with few time steps. The `makeTable` function produces a detailed log of the hedging policy, which is presented in Table 3.1. For each time step, the table show:

- The stock price,
- the option delta,
- the option value,
- the value of the replicating portfolio and the short bond position in that portfolio.

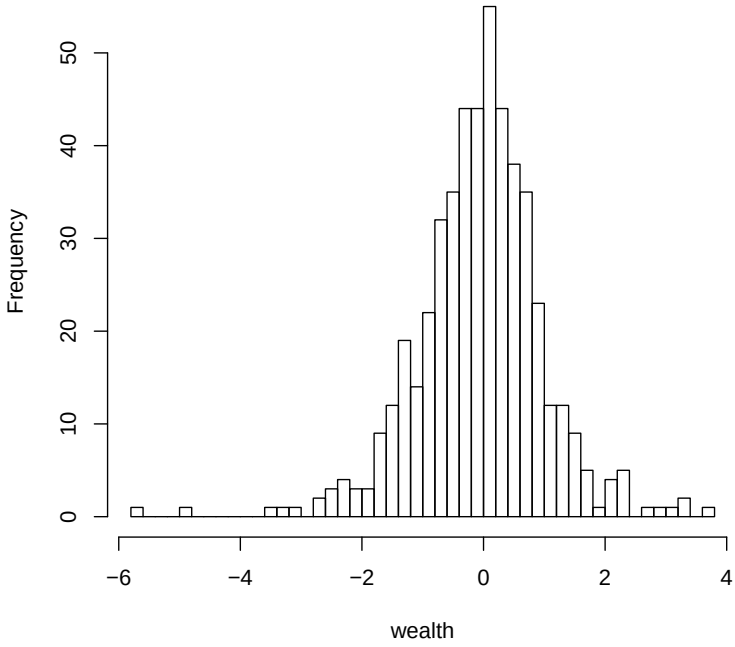



FIGURE 3.2: Distribution of hedging error when delta-hedging a European call with the Black-Scholes model. Option maturity: 1 year, volatility: 30%, interest rate: 2%.

```
> dtSim <- time(tSpot)[seq(1, dim(tSpot)[1], 10)]
> res <- deltaHedge(assets, sce.env, params = list(dtSim = dtSim,
      transaction.cost = 0), trace = F)
> xj <- makeTable(1, res)
```

*Design Considerations*

Two design features are worth mentioning.
The generation of the scenarios is independent from the expression of the dynamic trading strategies. Remember that every data element stored in a `DataProvider` is a `timeSeries`. Since all the calculations on `fInstrument` are vectorized, there is no difference between performing a calculation on a scalar, or performing a simulation on multiple scenarios.
The second aspect is the use of parent/child relationships among `DataProvider` objects. All the market data that is held constant in the simu-

| | time | stock price | delta | option | bond pos | hedge port. |
|---|---|---|---|---|---|---|
| 1 | 1.00 | 100.00 | 0.59 | 12.82 | -46.68 | 12.82 |
| 2 | 2.00 | 89.31 | 0.42 | 6.74 | -31.97 | 6.47 |
| 3 | 3.00 | 90.75 | 0.43 | 6.72 | -32.83 | 7.02 |
| 4 | 4.00 | 76.20 | 0.18 | 1.71 | -13.53 | 0.64 |
| 5 | 5.00 | 72.35 | 0.11 | 0.81 | -8.16 | -0.09 |
| 6 | 6.00 | 68.35 | 0.05 | 0.28 | -4.03 | -0.55 |
| 7 | 7.00 | 65.66 | 0.02 | 0.08 | -1.94 | -0.69 |
| 8 | 8.00 | 59.19 | 0.00 | 0.00 | -0.88 | -0.81 |
| 9 | 9.00 | 61.27 | 0.00 | 0.00 | -0.83 | -0.81 |
| 10 | 10.00 | 65.57 | 0.00 | 0.00 | -0.82 | -0.82 |
| 11 | 11.00 | 60.49 | 0.00 | 0.00 | -0.82 | -0.82 |

Table 3.1: Delta-hedging of a Europen call, strike: 100

lation is stored in the parent `DataProvider`. The data that changes from simulation to simulation is stored in the child `DataProvider`, and this is the object that is passed to the simulator. When a piece of data is requested from the child `DataProvider`, the following logic is applied:

1. First look for the data in the current `DataProvider` (the object passed as argument to the simulator)
2. if not found, look for the data in the parent `DataProvider`.
3. and so forth: the logic is recursive.

This behavior is inherited from the built-in `environment`.

# Part II

# Basic Fixed Income Instruments

# CHAPTER 4

# ELEMENTARY FIXED INCOME CALCULATIONS

*An honest man's word is as good as his bond*
Miguel de Cervantes

This chapter provides a elementary survey of fixed income calculations, with the aim of developping basic intuition regarding the relationships between the price and the yield of fixed income instruments.

## 4.1 BOND PRICING

In a simplified framework with even intervals between payments, the present value of a bond with coupon rate $c$ and maturing in $n$ years is given by

$$P(r) = \sum_{i=1}^{n} \frac{cN}{(1+r)^i} + \frac{N}{(1+r)^n} \tag{4.1}$$

where:

$c$ coupon rate, in decimal form

$N$ nominal

$n$ maturity, measured in years

$r$ yield, in decimal form

The formula can be implemented in one line:

```
> SimpleBondPrice <- function(coupon, n, N = 100, yield) {
    N * (sum(coupon/(1 + yield)^(1:n)) + 1/(1 + yield)^n)
 }
```

As an illustration, consider a bond with 3 years to maturity, a 5% coupon rate. With yield at 4%, price per 100$ nominal amount is

```
> P <- SimpleBondPrice(coupon = 0.05, n = 3, yield = 0.04)
```

or $P = 102.78$.

*The Price-Yield Curves*

Figure 4.1 represents the price-yield relationship for 20 year bonds with coupons ranging from 0 to 15%. In addition to the obvious non-linear nature of the price-yield relationship, it is worth noting that the convexity of the price-yield curve increases with the coupon rate.

```
> cp <- c(0, 0.05, 0.1, 0.15)
> cl <- c("red", "green", "blue", "black")
> y <- seq(0, 0.2, 0.01)
> n <- 20
> p <- matrix(0, length(cp), length(y))
> for (i in seq_along(cp)) {
      p[i, ] <- sapply(y, function(y) SimpleBondPrice(cp[i], n,
          yield = y))
  }
> plot(y, p[1, ], type = "l", lwd = 2, xlab = "Yield", ylab = "Price",
      col = cl[1], ylim = c(min(p), max(p)))
> for (i in 2:4) {
      lines(y, p[i, ], type = "l", lwd = 2, col = cl[i])
  }
> legend("topright", paste("cp = ", cp * 100, sep = ""), cex = 0.8,
      lwd = 2, bty = "n", col = cl)
```

Figure 4.2 represents the price-yield relationship for 10% bonds with maturities ranging from 3 to 30 years. Regardless of maturity, the bonds are worth 100 when the yield is identical to the coupon. This can be easily established by writing equation (4.1) as:

$$P(r) = N\left[\frac{c}{r}\left(1 - \frac{1}{(1+r)^n}\right) + \frac{1}{(1+r)^n}\right] \tag{4.2}$$

from which one concludes that $P(r) = N$ when $c = r$.

```
> m <- c(3, 5, 10, 30)
> cl <- c("red", "green", "blue", "black")
> y <- seq(0, 0.2, 0.01)
> cp <- 0.1
> p <- matrix(0, length(m), length(y))
> for (i in seq_along(m)) {
      p[i, ] <- sapply(y, function(y) SimpleBondPrice(cp, m[i],
          yield = y))
  }
> plot(y, p[1, ], type = "l", lwd = 2, xlab = "Yield", ylab = "Price",
      col = cl[1], ylim = c(min(p), max(p)))
> for (i in 2:4) {
```

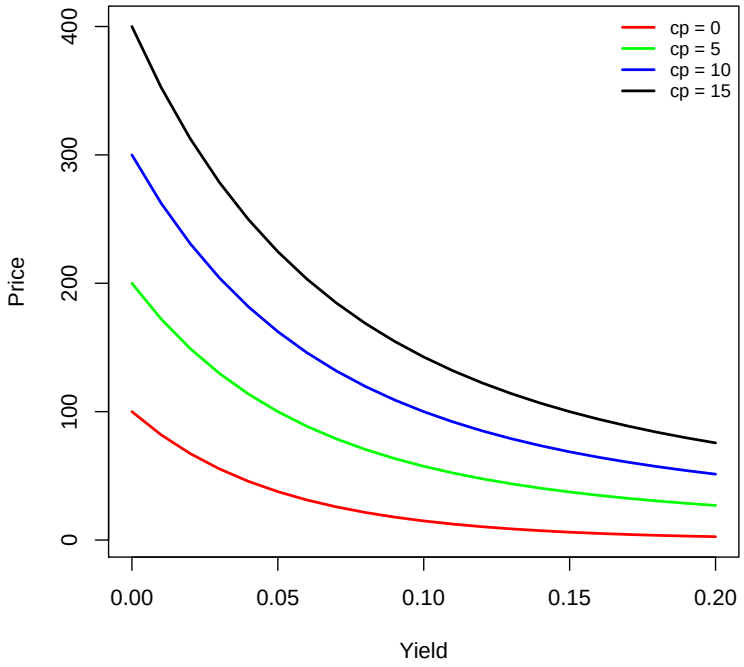



FIGURE 4.1: Price-Yield curves as a function of coupon rate. Bond maturity is 20 years.

```
    lines(y, p[i, ], type = "l", lwd = 2, col = cl[i])
}
> legend("topright", paste("m = ", m, sep = ""), cex = 0.8, lwd = 2,
    bty = "n", col = cl)
```

## 4.2 BASIC MEASURES OF INTEREST RATE RISK

The interest rate risk of fixed income instruments is traditionally measured by Variation, Sensitivity and Duration.

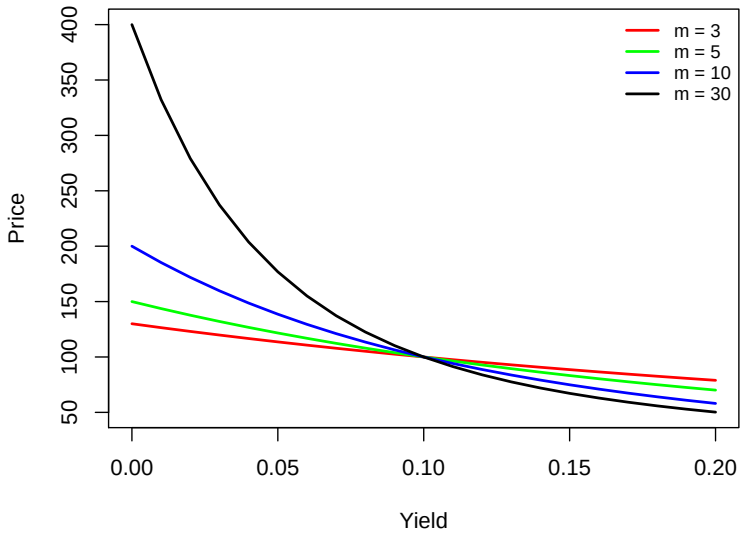



FIGURE 4.2: Price-Yield curves as a function of maturity. Bond coupon rate is 10%

*Variation*

Variation $V$ (also known as dollar duration) is the negative of the derivative of price with respect to yield

$$\begin{aligned} V &= -\frac{\partial P(r)}{\partial r} \\ &= \sum_{i=1}^{n} \frac{i F_i}{(1+r)^{i+1}} \end{aligned}$$

```
> Variation <- function(coupon, n, N = 100, yield) {
    100 * sum(sapply(1:n, function(i) ifelse(i < n, i * coupon/(1 +
        yield)^(i + 1), n * (1 + coupon)/(1 + yield)^(n + 1))))
}
```

Variation is often measured by "PV01", which is the change in bond price for a 1 basis point change in yield:

$$\text{PV01} = V \times 10^{-4}$$

| Bond | Coupon (%) | Maturity | Quantity | Yield (%) |
|---|---|---|---|---|
| A | 5 | 5 | 2 | 5.20 |
| B | 6 | 7 | 3 | 5.60 |

Table 4.1: Bond portfolio

*Approximate Portfolio Yield*

Variation is a useful tool for computing the approximate yield of a portfolio, given the yield of each bond. Consider two bonds with prices $P_1(r_1)$ and $P_2(r_2)$ and variations $V_1(r_1)$ and $V_2(r_2)$. We look for the common yield $r^*$ such that:

$$P = a_1 P_1(r^*) + a_2 P_2(r^*) \tag{4.3}$$

Use a first-order Taylor's expansion of price $P_i(r)$ at $r_i$:

$$P_i(r^*) = P_i(r_i) + (r^* - r_i) V_i(r_i)$$

Replacing $P_i(r^*)$ by this approximation in (4.3) yields:

$$\sum_{i=1}^{2} a_i (r^* - r_i) V_i(r_i) = 0$$

which gives

$$r^* = \frac{\sum_{i=1}^{2} a_i r_i V_i(r_i)}{\sum_{i=1}^{2} a_i V_i(r_i)}$$

This approximation is valid when the yields $r_i$ are not too far apart, as illustrated by the following calculation.
Consider a portfolio made of two bonds, described in Table 4.1.
The approximate portfolio yield is computed as follows:

```
> cp <- c(0.05, 0.06)
> r <- c(0.052, 0.056)
> T <- c(5, 7)
> a <- c(2, 3)
> P <- sapply(1:2, function(i) SimpleBondPrice(cp[i], T[i], yield = r[i]))
> V <- sapply(1:2, function(i) Variation(cp[i], T[i], yield = r[i]))
> r <- sum(r * V * a)/sum(V * a)
```

which gives an approximate yield of $r = 5.47\%$. An exact calculation involves solving for $r^*$ the non-linear equation

$$P = \sum_{i=1}^{2} a_i P_i(r^*)$$

```
> fun <- function(r) {
     sum(P) - sum(sapply(1:2, function(i) SimpleBondPrice(cp[i],
         T[i], yield = r)))
 }
> r <- uniroot(fun, c(0.05, 0.06))$root
```

which gives the exact portfolio yield $r = 5.43\%$.

*Sensitivity*

The sensitivity $S$ (also called modified duration) is the percent change in price per unit change in yield:

$$S = \frac{1}{P} V = \frac{1}{P(1+r)} \sum_{i=1}^{n} \frac{i F_i}{(1+r)^i}$$

```
> Sensitivity <- function(coupon, n, N = 100, yield) {
     Variation(coupon, n, N, yield)/SimpleBondPrice(coupon, n,
         N, yield)
 }
```

*Modified Duration of a Portfolio*

Consider a portfolio made of $n$ bonds, each with nominal amount $q_i$, price $P_i$ and modified duration $S_i$, $i = 1, \ldots, n$. Let $P$ be the value of the portfolio, we want to determine the modified duration of the portfolio

$$S = \frac{1}{P} \frac{\partial P}{\partial r}$$

$$\begin{aligned} S &= \frac{1}{P} \sum_i q_i \frac{\partial P_i}{\partial r} \\ &= \frac{1}{P} \sum_i q_i \frac{P_i}{P_i} \frac{\partial P_i}{\partial r} \\ &= \frac{1}{P} \sum_i q_i P_i S_i \\ &= \sum_i \frac{q_i P_i}{P} S_i \end{aligned} \tag{4.4}$$

The modified duration of a portfolio is the weighted average of the modified durations of the bonds. The weight associated with each bond is the fraction of portfolio value associated with the bond.

*Duration*

The Macaulay duration $D$ (named after Frederick Macaulay who introduced this risk measure), can be interpreted as the average maturity, weighted by the present value of the cash flow at each payment date.

$$D = \sum_{i=1}^{n} \frac{i F_i}{P(1+r)^i} \tag{4.5}$$

The Macaulay Duration is closely related to Sensitivity:

$$S = \frac{1}{(1+r)} D$$

```
> SimpleDuration <- function(coupon, n, yield) {
      100 * sum(sapply(1:n, function(i) ifelse(i < n, i * coupon/(1 +
          yield)^i, n * (1 + coupon)/(1 + yield)^n)))/SimpleBondPrice(coupon,
          n, yield = yield)
  }
```

These measures can be used to estimate the approximate impact of yield change on price. Using the previous example, the exact price change caused by a 0.1% yield increase is:

```
> d1 <- SimpleBondPrice(coupon = 0.06, n = 10, yield = 0.051)
> -SimpleBondPrice(coupon = 0.06, n = 10, yield = 0.05)
[1] -107.72
```

or $d_1 = 106.92$.
An approximate calculation with Duration yields:

```
> P <- SimpleBondPrice(coupon = 0.06, n = 10, yield = 0.05)
> D <- SimpleDuration(coupon = 0.06, n = 10, yield = 0.05)
> d2 <- -P * D/(1 + 0.05) * 0.001
```

or a change in price of $d_2 = -0.81$ for a .1% increase in yield. Duration is a popular measure of price sensitivity because of its intuitive interpretation: the price sensitivity of a bond with duration $D$ is the same as the one of a zero-coupon bond maturing in $D$ years.
It is a surprising fact that the duration of coupon bonds does not increase indefinitely as maturity increases. The following script computes the duration of bonds of varying maturities and coupon rates.

```
> coupon <- c(0, 0.1, 0.2, 2, 10, 20)/100
> maturity <- seq(from = 1, to = 100, by = 1)
> yield <- 0.1
> res <- NULL
> for (c in coupon) {
      res <- rbind(res, sapply(maturity, function(n) SimpleDuration(c,
          n, yield)))
  }
```

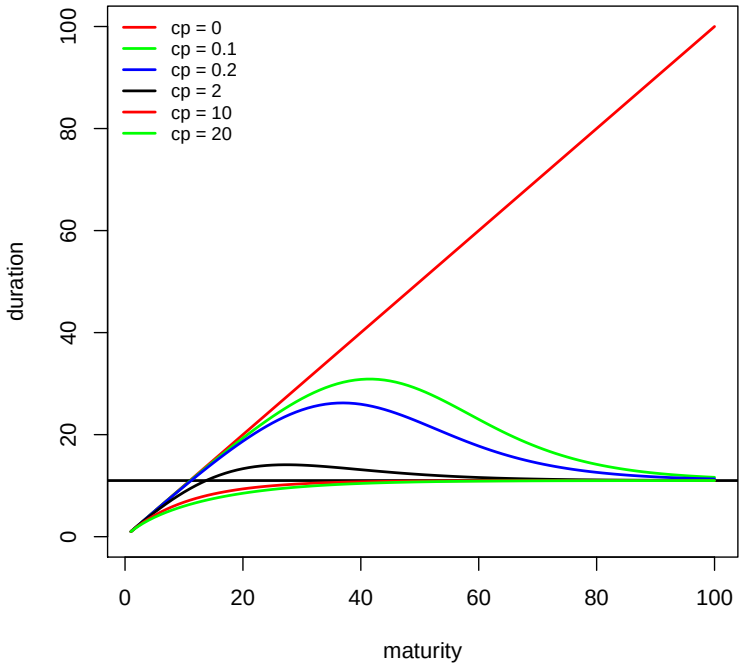



FIGURE 4.3: Duration of a bond as a function of its maturity, when yield = 10%.

The resulting values are plotted in Figure 4.3. As expected, the duration of the zero-coupon bond is identical to its maturity. For all other bonds, the duration converges to a constant as maturity increases, regardless of coupon rate.
This can be formally shown by first deriving an analytical formula for Duration. Let $c$ be the coupon rate and $r$ the bond yield. The price of a bond with nominal of 1 is:

$$\begin{aligned} P &= \frac{c}{r}(1-(1+r)^{-n})+(1+r)^{-n} \\ &= \frac{1}{r}\left(c(1-(1+r)^{-n})+(1+r)^{-n}\right) \\ &= \frac{1}{r}g(r) \end{aligned}$$

with $g(r)=c(1-(1+r)^{-n})+(1+r)^{-n}$.

Now,

$$\begin{aligned}\frac{\partial \log P}{\partial r} &= -\frac{1}{r}+\frac{g'(r)}{g(r)}\\ &= \frac{1}{P}\frac{\partial P}{\partial r}\end{aligned}$$

But,

$$\begin{aligned}D &= -\frac{1+r}{P}\frac{\partial P}{\partial r}\\ &= (1+r)\left[\frac{1}{r}-\frac{g'(r)}{g(r)}\right]\\ &= \frac{1+r}{r}-(1+r)\frac{g'(r)}{g(r)}\\ &= \frac{1+r}{r}-(1+r)\frac{n(c-r)(1+r)^{-1}+1}{c((1+r)^n-1)+r}\\ &= \frac{1+r}{r}-\frac{n(c-r)+1+r}{c((1+r)^n-1)+r}\end{aligned}$$

The last expression shows that duration converges towards $\frac{1+r}{r}$ as maturity goes to infinity.

*Duration of a Portfolio*

Consider a portfolio made of $n$ bonds, each with nominal amount $q_i$, price $P_i$ and duration $D_i, i = 1,\ldots,n$. Let $P$ be the value of the portfolio, what is its Duration? Since Duration is related to sensitivity by the relation

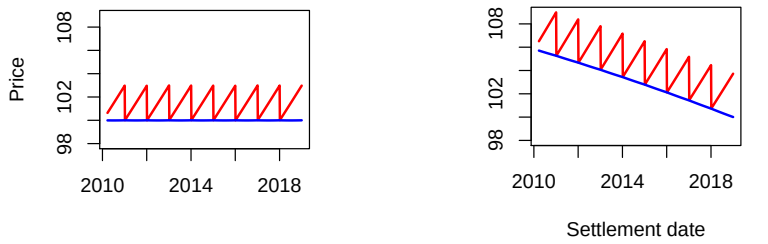
$$D = (1+r)S$$

multiplying both sides of equation 4.4 by $(1+r)$ yields:

$$D = \sum_i \frac{q_i P_i}{P} D_i \tag{4.6}$$

## 4.3 BOND CLEAN AND DIRTY PRICE

Bonds prices are quoted in "flat" or "clean" price, and not in net present value. The clean price being the net present value less the accrued interest. Therefore, the bond quoted price is not the amount that the buyer disburses when she buys a bond. The reason for this apparent complication is illustrated in Figure 4.4, which plots the change over time of a bond clean price and present value, with yield held constant. Quoting a bond by net present value would be impractical, because of the discontinuity around the coupon payment date. The clean price, however, remains steady over time, when yield is held constant.




Par bond (coupon=yield) Premium bond (coupon>yield)

FIGURE 4.4: Clean price (blue) and NPV (red) of bonds as a function of time, for yield held constant at 3%. On coupon date, the NPV drops by the coupon amount, but the clean price remains steady.

The plots are obtained as follows, using the `Bond` function of package `empfin`:

```
> b375 <- Bond("375", myDate("01jan2000"), myDate("04jan2019"),
     0.0375, 100, "a")
> b300 <- Bond("300", myDate("01jan2000"), myDate("04jan2019"),
     0.03, 100, "a")
> dtToday <- myDate("25mar2010")
> dt <- timeSequence(dtToday, myDate("04jan2019"), by = "day")
> dt <- dt[seq(1, length(dt), by = 4)]
> bondcalc <- function(bond, y) {
     p <- sapply(dt, function(d) BondYield2Price(bond, d, y))
     ac <- rep(NA, length(p))
     for (i in seq(length(p))) {
         ac[i] <- BondAC(bond, as.Date(dt[i]))
     }
     cp <- p - ac
     list(p = p, ac = ac, cp = cp)
 }
> bc1 <- bondcalc(b300, 0.03)
> bc2 <- bondcalc(b375, 0.03)
```

# CHAPTER 5

# THE TERM STRUCTURE OF INTEREST RATES

```
> library(fBasics)
> library(empfin)
> data(LiborRates)
> library(YieldCurve)
> data(FedYieldCurve, package = "YieldCurve")
> data(ECBYieldCurve, package = "YieldCurve")
> data(LiborRates, package = "empfin")
```

The relationship between yield and maturity is called the "term structure of interest rate". Since there are many ways to measure yield, there are also many types of term structures. The term structure of interest rate is important because, everything else being equal, maturity is the main determinant of yield. In addition, investors and economists believe that the shape of this curve may reflect the market's expectation of future rates.

## 5.1 TERM STRUCTURE SHAPES

The term structure of interest rates can assume a variety of shapes, as illustrated by the following 2 examples.
Figure 5.1 displays a sample of yield curves, published by the European Central Bank (ECB). These curves represent the yield to maturity for AAA rated euro-zone government bonds. We show quarterly data from December 2006 to June 2009. Three patterns are visible:

- an almost flat curve (August 2007)
- an inverted curve at the short end, and upward sloping at the long end (January 2008)
- an evenly upward-sloping curve (December 2006, June 2009)

The data is extracted from the `YieldCurve` package, data set `ECBYieldCurve`:

```
> tau <- c(3/12, 6/12, 1:30)
> d1 <- as.Date("2006-12-29")
> d2 <- as.Date("2009-07-24")
> nbObs <- dim(ECBYieldCurve)[1]
> dtObs <- seq(d1, d2, length.out = nbObs)
> indx <- seq(1, nbObs, 90)
> YC <- ECBYieldCurve[indx, ]
> dtObs <- dtObs[indx]
> nbSample <- length(indx)
```

The plot is generated by:

```
> mat = dtObs[1] + (tau * 365)
> plot(mat, YC[1, ], type = "l", ylim = c(0, 5), ylab = "Euro Sovereign Bond Yield",
      col = 1, lwd = 2)
> for (i in seq(2, nbSample)) {
      mat = dtObs[i] + (tau * 365)
      lines(mat, YC[i, ], type = "l", col = i, lwd = 2)
  }
> legend("bottomright", legend = as.character(dtObs), col = seq(1,
      nbSample), lty = 1, lwd = 2)
```

Figure 5.2 presents a similar plot for the US Treasury yield, over the period 1982-2009. This illustrates the wide range of slope that a yield curve can assume over a long period of time. The plot uses the data set `FedYieldCurve`, and is obtained as follows:

```
> tau <- c(3/12, 6/12, 1, 5, 7, 10)
> dtObs <- as.Date(timeSequence(from = "1982-01-01", to = "2009-01-01",
      by = "month"))
> nbObs <- dim(FedYieldCurve)[1]
> indx <- seq(1, nbObs, 12)
> YC <- FedYieldCurve[indx, ]
> dtObs <- dtObs[indx]
> nbSample <- length(indx)
> mat = dtObs[1] + (tau * 365)
> plot(mat, YC[1, ], type = "l", ylim = c(0, 15), xlim = c(dtObs[1],
      dtObs[nbSample] + 10 * 365), col = 1, lwd = 2, ylab = "Treasury Yield")
> for (i in seq(2, nbSample)) {
      mat = dtObs[i] + (tau * 365)
      lines(mat, YC[i, ], type = "l", col = i, lwd = 2)
  }
```

## 5.2 BUILDING A ZERO-COUPON BOND CURVE

*Principle*

In a stylized manner, the calculation of a bond zero-coupon yield curve is straight-forward. Assume a set of bonds with cash flows $F_{i,j}$ occurring at regular intervals, where $i$ is the index of the bond and $j$ the index of the

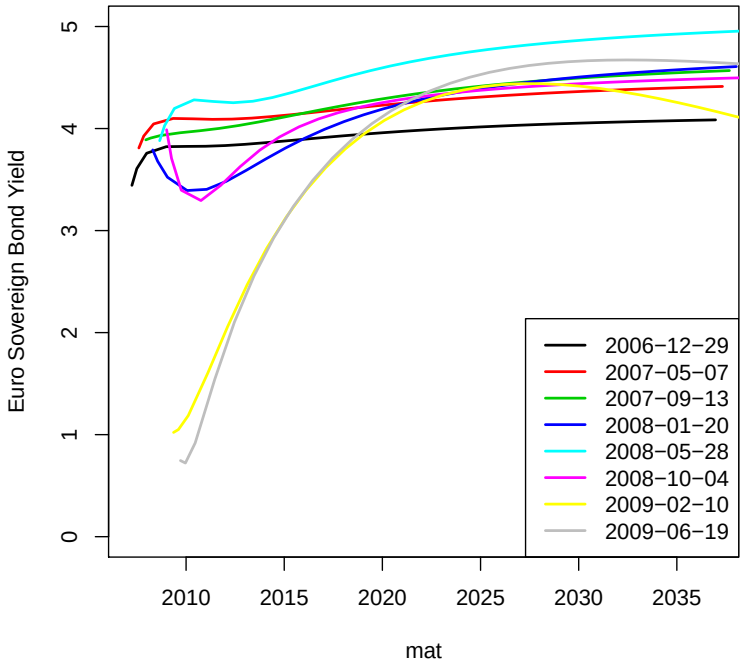



FIGURE 5.1: The yield to maturity curves of Euro-zone AAA government bonds

cash flow date. We look for an interest rate function $z(t)$ that prices the bonds exactly:

$$P_i = \sum_{j=1}^{n_i} F_{i,j}\, e^{-z(t_j)t_j}$$

The function $z(t)$ is called the continuously compounded zero-coupon yield curve, or "continuous zero curve" in short.
How to perform the calculation is best explained with an example. Assume that we observe 4 bonds, as summarized in Table 5.1.
Let $y_i$ be the bond yield for bond $i$ maturing in $i$ years, and $P_i$ the corresponding present value.

```
> y <- c(5.5, 6, 7.5, 8.5)/100
> c <- c(9, 7, 5, 4)/100
> P <- sapply(seq(4), function(i) SimpleBondPrice(c[i], i, y[i]))
> z <- rep(0, 4)
> r <- rep(0, 4)
```

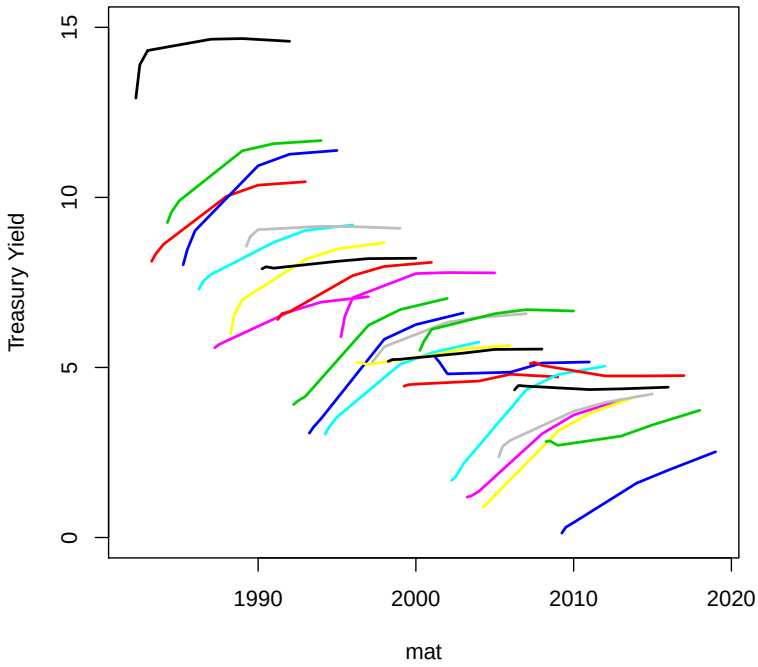



FIGURE 5.2: The yield to maturity curves of US government bonds

| Maturity | Coupon (%) | Bond Yield (%) |
|---|---|---|
| 1 | 9 | 5.5 |
| 2 | 7 | 6.0 |
| 3 | 5 | 7.5 |
| 4 | 4 | 8.5 |

Table 5.1: Bond data for calculating a Treasury Zero-Coupon Curve

The zero-coupon rates $z_i$ for each maturity are computed recursively, one maturity at a time. The one year zero-rate is obtained by converting the bond yield into a continuously compounded rate:

$$e^{-z_1} = \frac{1}{(1+y_1)}$$

or,

$$z_1 = \ln(1+y_1)$$

which gives $z_1 = 5.35\%$.
The zero coupon rate for year $i$, with the rates up to year $i-1$ assumed known, is obtained by solving for $z_i$:

$$P_i = 100\left(\sum_{j=1}^{i-1} c e^{-jz_j} + (1+c)e^{-iz_i}\right)$$

which yields:

$$z_i = -\frac{1}{i}\log\left(\frac{1}{1+c}\left(\frac{P_i}{100} - c\sum_{j=1}^{i-1} e^{-jz_j}\right)\right)$$

This recursive calculation is performed by the function `ZCRate`:

```
> ZCRate <- function(c, z, i) {
      zc <- -(1/i) * log(1/(1 + c[i]) * (P[i]/100 - c[i] * sum(exp(-seq(i -
          1) * z[1:(i - 1)]))))
      zc
  }
> for (i in seq(2, 4)) {
      z[i] <- ZCRate(c, z, i)
  }
```

A third term structure of interest is the par yield curve. For a given maturity, the par yield is the coupon rate of a bond priced at par (i.e. priced at 100%). Formally, given a zero-coupon curve, the par yield $r_i$ for maturity $i$ is such that:

$$1 = r_i \sum_{j=1}^{i} e^{-jz_j} + e^{-iz_i}$$

Back to our example, the par yields are computed by the following function, noting that the one year par rate is by definition the corresponding bond yield:

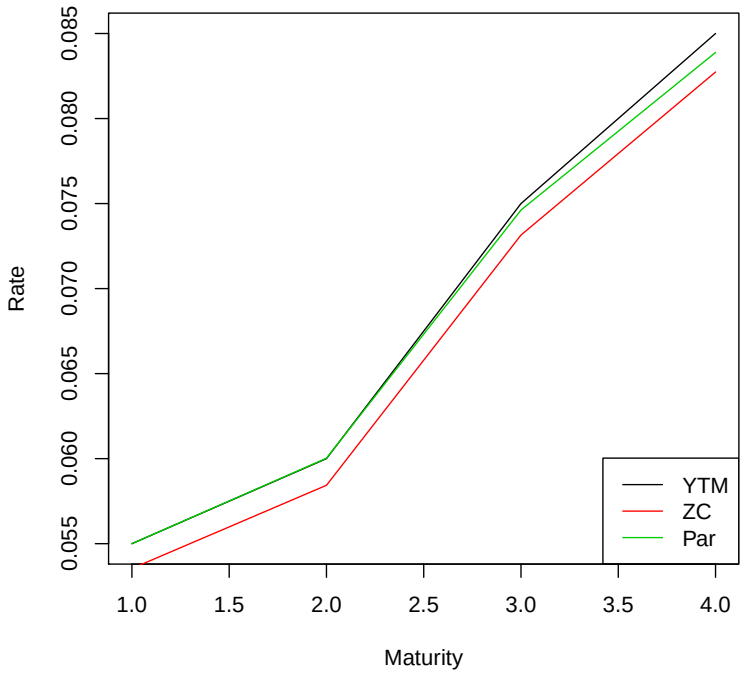



FIGURE 5.3: The yield to maturity curve and the corresponding zero-coupon and par curves

```
> ParRate <- function(i) {
      r <- (1 - exp(-i * z[i]))/sum(exp(-seq(i) * z[1:i]))
      r
  }
> r[1] <- y[1]
> for (i in seq(2, 4)) {
      r[i] <- ParRate(i)
  }
```

Figure 5.3 displays the bond yield curve and the corresponding zero-coupon curve and par curves.
In practice, this simple method is not realistic for several reasons:

- In most markets, we will not find a set a bonds with identical anniversary dates,
- the method outlined above only provides values of $z(t)$ at maturity dates corresponding to the cash flow dates. An interpolation method is needed to obtain $z(t)$ at arbitrary maturities, and finally

- it does not account for the fact that some bonds may trade at a premium or discount: for example, a bond with a small outstanding amount will often trade at a discount, while the current 10 year benchmark bond will trade at a premium.

If we insist on a zero-coupon curve that prices exactly every bond in the sample, we will get a unnatural shape for the zero curve, and even more so for the forward rates.
The alternative is to specify a functional form (sufficiently flexible) for the zero-coupon curve, and to estimate its parameters by minimizing the pricing error over a set of bonds.
Since the term structure of interest rate can equivalently be represented in terms of spot rates, forward rates or discount factors, one has three choices for specifying the functional form of the term structure.

*Functional forms for the zero-coupon curve*

**Modeling the instantaneous forward rate** A popular method is to model the instantaneous forward rate, $f(t)$. Integrating this function gives the zero-coupon rate. Nelson & Siegel (1987) introduced the following model for the instantaneous forward rate:

$$f(t) = \beta_0 + \beta_1 e^{-\frac{t}{\tau_1}} + \beta_2 \frac{t}{\tau_1} e^{-\frac{t}{\tau_1}}$$

Integrating $f(t)$, the corresponding zero-coupon rates are given by:

$$z(t) = \beta_0 + \beta_1 \frac{1 - e^{-\frac{t}{\tau_1}}}{\frac{t}{\tau_1}} + \beta_2 \left( \frac{1 - e^{-\frac{t}{\tau_1}}}{\frac{t}{\tau_1}} - e^{-\frac{t}{\tau_1}} \right)$$

Svensson (1994) extended this model with 2 additional parameters, defining the instantaneous forward rate as:

$$f(t) = \beta_0 + \beta_1 e^{-\frac{t}{\tau_1}} + \beta_2 \frac{t}{\tau_1} e^{-\frac{t}{\tau_1}} + \beta_3 \frac{t}{\tau_2} e^{-\frac{t}{\tau_2}}$$

The following script, taken from the `YieldCurve` package, illustrates the fitting of the Nelson-Siegel and Svensson models to US Treasury yields.

```
> tau <- c(3, 6, 12, 60, 84, 120)
> mediumTerm <- c(12, 60, 84)
> Parameters.NS <- Nelson.Siegel(rate = FedYieldCurve[5, ], maturity = tau,
    MidTau = mediumTerm)
> Parameters.S <- Svensson(FedYieldCurve[5, ], tau, c(3, 12), c(60,
    120))
> tau.sim <- seq(1, 120, 2)
> y.NS <- NSrates(Parameters.NS[1, 1:3], Parameters.NS$lambda[1],
    tau.sim)
> y.S <- Srates(Parameters.S[1, 1:4], Parameters.S[1, 5:6], tau.sim,
    whichRate = "Spot")
```

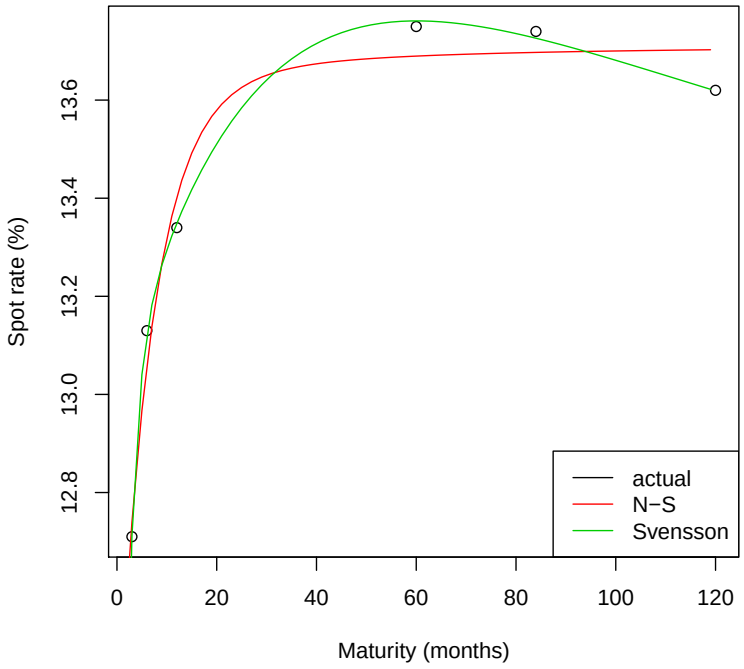



FIGURE 5.4: Fitted Treasury zero-coupon spot curves, with model of Svensson and Nelson-Siegel.

Figure 5.4 shows a comparison of the two fitted spot curves.
The corresponding instantaneous fitted forward curves are displayed in figure 5.5.

```
> f.S <- Srates(Parameters.S[1, 1:4], Parameters.S[1, 5:6], tau.sim,
     whichRate = "Forward")
> forward.NS <- function(beta, lambda, tau) {
     beta <- data.matrix(beta)
     MoverTau <- lambda * tau
     eMoT <- exp(-MoverTau)
     beta[1] * rep(1, length(tau)) + beta[2] * eMoT + beta[3] *
         MoverTau * eMoT
 }
> f.NS <- forward.NS(Parameters.NS[1, 1:3], Parameters.NS$lambda[1],
     tau.sim)
```

The Svensson or Nelson-Siegel curves can be easily decomposed into 3 elementary shapes that capture the main features of a yield curve: level,

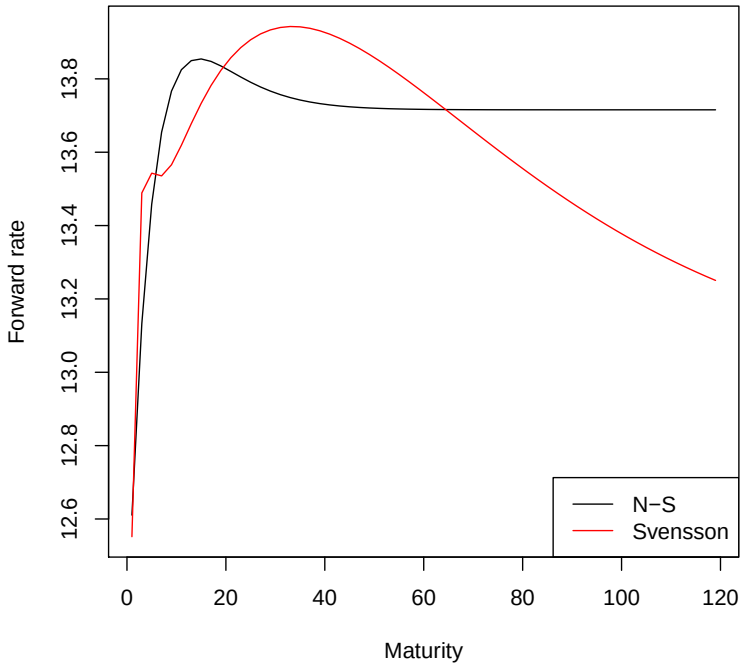



FIGURE 5.5: Fitted Treasury Zero-coupon forward curves, with Svensson model and Nelson-Siegel model

slope and curvature. Figure 5.6 shows the decomposition of the Svensson fitted spot curve into these three components.

**Empirical justification** The functional forms chosen by Svensson or Nelson and Siegel are comforted by empirical observations. Indeed, a principal components analysis of daily changes in zero-coupon rates shows that actual changes are accounted for by three factors that have shapes consistent with the decomposition illustrated in Figure 5.6.

The data set consists in AAA euro-zone government rates, observed from December 29, 2006 to July 24, 2009.

```
> n <- 252
> daily.zc.change <- ECBYieldCurve[2:n, ] - ECBYieldCurve[1:(n -
      1), ]
> pca <- princomp(daily.zc.change, center = TRUE, scale = TRUE)
```

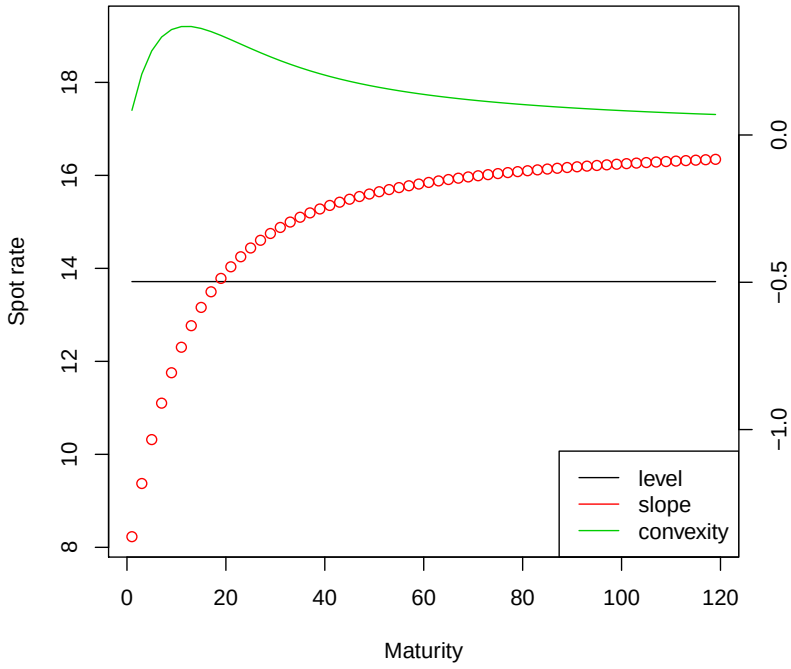



FIGURE 5.6: Components of Fitted Treasury zero-coupon spot curves, with Nelson-Siegel model

Figure 5.7 shows the contribution of the first 6 factors to the total variance, and shows that of the bulk of the variance (about 90%) is captured by the first 3 factors.

The factor loadings show the sensitivity of zero-coupon of various maturities to factor movements. The first three factors are displayed in Figure 5.8.

Focusing on the shapes for maturities over 2 years, the first factor (red) can be interpreted as a level factor, since it affects all zero-coupons in a similar way (except for the short maturities). The second factor (blue) can be interpreted as a slope or rotation factor. Finally, the third factor (green) can be interpreted as a curvature factor. We find a good qualitative agreement with the decomposition of Figure **??**.

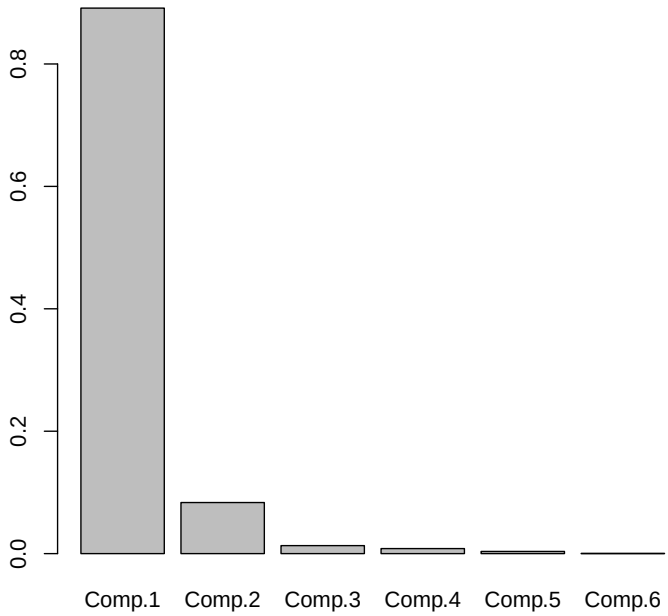



FIGURE 5.7: Contribution (in %) of the first 6 principal components to total variance (norm of the covariance matrix of changes in zero-rates).

## 5.3 BOOTSTRAPPING A ZERO-COUPON LIBOR CURVE

Given a set of LIBOR deposit rates and swap rates, we would like to compute the zero-coupon curve $z_i$. Again, this is best explained by an example. We use the `LiborRates` data set, which contains a time series of deposit rates and swap rates for various maturities. The construction of this data set was described in Chapter 1. The data for November 10, 2009, is displayed in Table 5.2.

### *Bootstrap from the deposit rates*

The first part of the calculation is simple, and amounts to converting the LIBOR deposit rates ($\frac{\text{ACT}}{360}$) into zero-coupon continuous rates ($\frac{\text{ACT}}{365}$), using the relation:

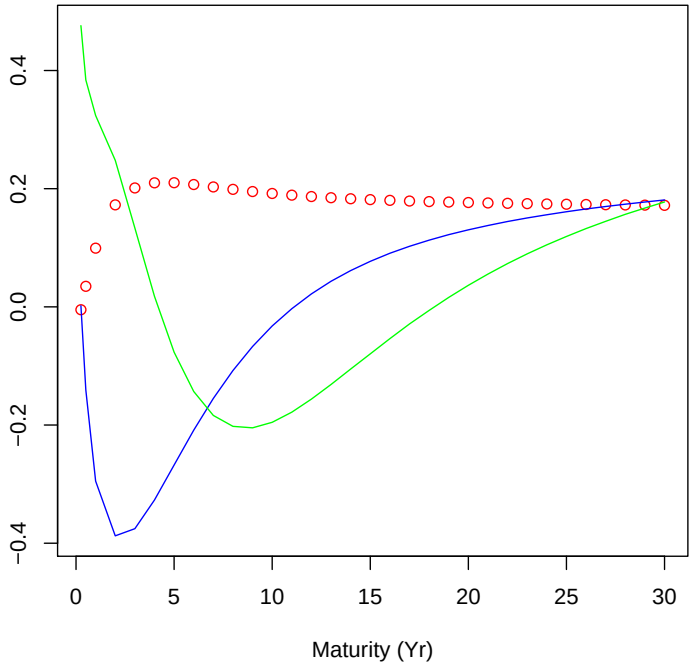



FIGURE 5.8: Factor loadings for first three principal components.

```
> dtToday <- as.timeDate("2009-11-10")
> l <- ts_libor[dtToday]/100
> print(l)
GMT
           Libor1M Libor3M Libor6M Swap1Y Swap2Y Swap3Y Swap4Y Swap5Y Swap7Y
2009-11-10   0.003  0.0045   0.007 0.0052 0.0117 0.0178 0.0225 0.0262 0.0315
           Swap10Y Swap30Y
2009-11-10  0.0359  0.0423
```

Table 5.2: One row of the LiborRates data set.

$$(1+\frac{d}{360}l)=e^{z\frac{d}{365}}$$

where:

$d$ Actual number of days from settlement date to maturity

$l$ Deposit rate

$z$ Zero-coupon rate

The 3 and 6 months zero-coupon rates are:

```
> dtToday <- as.timeDate("2009-11-10")
> dtPayment <- timeSequence(from = dtToday, by = "3 months", length.out = 2)
> z.deposits = rep(0, 2)
> dd = as.numeric(dtPayment[1] - dtToday)
> r <- as.numeric(l[, "Libor3M"])
> z.deposits[1] = log(1 + r * dd/360) * 365/dd
> dd = as.numeric(dtPayment[2] - dtToday)
> r <- as.numeric(l[, "Libor6M"])
> z.deposits[2] = log(1 + r * dd/360) * 365/dd
```

*Bootstrap from the swap rates*

At one year and beyond, the calculation proceeds recursively, one year at a time. The swap rate is a par yield, meaning that the fixed leg of a swap has a PV of 100. We can use the same logic as in the previous section, and compute recursively the zero-coupon rate for each maturity.
The calculation is performed by the following script, where we linearly interpolate the swap rates for the missing maturities.
The linear interpolation is performed with:

```
> l.mat <- c(1, 2, 3, 4, 5, 7, 10)
> l.c <- l[4:10]
> mat <- seq(10)
> c.libor <- approx(l.mat, l.c, xout = mat)$y
```

The recursive computation of the zero-coupon rates is finally done with:

```
> z.libor <- rep(0, 10)
> z.libor[1] <- log(1 + c.libor[1])
> P <- rep(100, length(mat))
> for (i in seq(2, 10)) {
      z.libor[i] <- ZCRate(c.libor, z.libor, i)
  }
```

The resulting zero-curve is the concatenation of the zero-coupon deposit rates and the zero-coupon swap rates. It is shown in figure 5.9

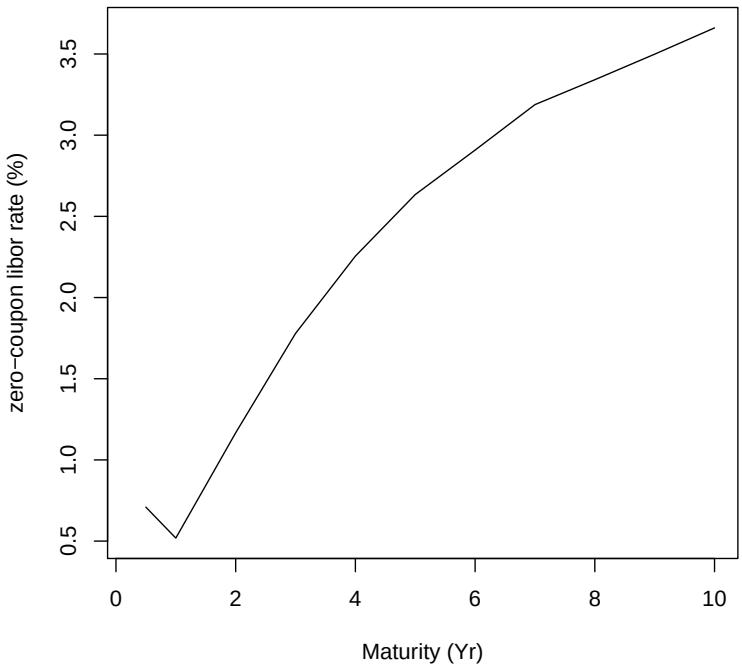



FIGURE 5.9: Zero-coupon curve bootstrapped from Libor deposits and swaps

# CHAPTER 6

# FIXED INCOME RISK MANAGEMENT

```
> library(linprog)
> library(xtable)
> library(empfin)
```

In this chapter, we introduce the two broad categories of Fixed Income risk management strategies: immunization and dedication. The problem being addressed is simple: given a liability stream (anticipated future payments, certain or contingent), and given a initial budget, how should the funds be invested so that the proceeds from the investment portfolio may fund the liability, under all foreseeable interest rate scenarios?
The immunization strategy is a two steps process. The manager first identifies the interest rate risk factors, and then constructs a portfolio such that assets and liabilities have the same exposure to the risk factors, therefore, the global position (assets - liabilities) is immunized against movements of the risk factors.
The dedication or cash flow matching strategy takes a different approach, but is conceptually simple. The goal of this strategy is to avoid interest rate risk by constructing a portfolio that will generate in a timely manner the cash flow needed to meet the liability, thereby evacuating interest risk by construction.
In practice, fund managers often use a mix of both strategies. The two strategies are next described and illustrated.

## 6.1 IMMUNIZATION

The method is best described by the following elementary example. A fund manager has a certain liability $L_T$ to be paid at time $T$. The present value of this liability is $L_T(1+r)^{-T}$. The manager wants to invest that budget in

| Bond | Coupon | Maturity |
|---|---|---|
| A | .05 | 3 |
| B | .06 | 7 |

Table 6.1: Characteristics of bonds in the immunization portfolio

a portfolio that will provide the wealth $L_T$ at the required time, regardless of future interest rates.
She constructs a portfolio that generates cash flows $F_i$ at time $T_i$. Cash flows paid at time $T_i < T$ will be reinvested till $T$, and those occurring after $T$ will be sold at $T$. The value of this portfolio at $T$ is thus:

$$V_T = \sum_i F_i(1+r)^{T-T_i}$$

and must be equal to $L_T$. Let's compute the Variation of $V_T$:

$$\begin{aligned}\frac{\partial V_T}{\partial r} &= \sum_i F_i(T-T_i)(1+r)^{T-T_i-1} \\ &= \left[\sum_i F_i T(1+r)^{-T_i} - \sum_i F_i T_i(1+r)^{T_i}\right](1+r)^{T-1} \\ &= V_T\left[T\frac{\sum_i F_i(1+r)^{-T_i}}{V_T} - \frac{\sum_i F_i T_i(1+r)^{T_i}}{V_T}\right](1+r)^{T-1}\end{aligned}$$

In the last expression, we recognize the definition of Duration (4.5), so that the Variation of $V_T$ is finally:

$$\frac{\partial V_T}{\partial r} = V_T[T-D](1+r)^{T-1}$$

where $D$ is the duration of the stream of cash flows $F_i$. This equation shows that if the stream of cash flows $F_i$ has the same Duration as the investment horizon $T$, then, in a first-order approximation, the wealth $V_T$ is insensitive to changes in yield. If yield deceases, the lower value of the reinvested cash flows will be offset by the higher present value of the cash flows maturing after $T$. The converse is true if yield increases. This investment strategy, which aims at preserving wealth at a given investment horizon is called immunization.
How to construct the assets portfolio is described next. Let the liability $V_T = 100$, $T = 5$, $V_0 = V_T(1+r)^{-T}$. Current yield is $r = .05$. The investment portfolio will be made of two bonds described in Table 6.1
We next determine the quantities $q_A$ and $q_B$ of each bond in the investment portfolio, so that the present value and the Duration of the portfolio

match those of the liability. Remember from section 4.2 that the Duration of a portfolio is a weighted average of the Durations of the bonds in the portfolio.
We express these two conditions by the following equations:

$$\begin{aligned} V_0 &= q_A P_A + q_B P_B \\ T &= q_A \frac{P_A D_A}{V_0} + q_B \frac{P_B D_B}{V_0} \end{aligned}$$

or, using matrix notation:

$$AQ = b$$

with

$$A = \begin{pmatrix} P_A & P_B \\ \frac{P_A D_A}{V_0} & \frac{P_B D_B}{V_0} \end{pmatrix} Q = \begin{pmatrix} q_A \\ q_B \end{pmatrix} b = \begin{pmatrix} V_0 \\ T \end{pmatrix}$$

This system is solved for $q_A$ and $q_B$ to determine the portfolio composition.

```
> r <- 0.05
> T <- 5
> V_0 <- 100 * (1 + r)^(-T)
> P_A <- SimpleBondPrice(0.05, 3, r)
> P_B <- SimpleBondPrice(0.06, 7, r)
> D_A <- SimpleDuration(0.05, 3, r)
> D_B <- SimpleDuration(0.06, 7, r)
> A <- matrix(c(P_A, P_B, D_A * P_A/V_0, D_B * P_B/V_0), nrow = 2,
    byrow = TRUE)
> b <- matrix(c(V_0, T), nrow = 2)
> Q = solve(A, b)
> q_A <- Q[1]
> q_B <- Q[2]
```

The solution is $q_A = 0.24, q_B = 0.51$. To verify the robustness of the selected portfolio, we next simulate the wealth at horizon $T$, under various scenarios for $r$.
Let $V_A$ (resp. $V_B$) be the wealth at time $T$ generated by 1 unit of bond A (resp. B).

$$V_A = \sum_i F_A(t_i)(1+r)^{(T-T_i)}$$

where $F_A(t)$ is the cash flow generated at time $t$ by one unit of bond $A$. The wealth at time $T$ is the compounded value of the cash flows occurring before $T$, and the discounted value of the cash flows occurring after $T$.

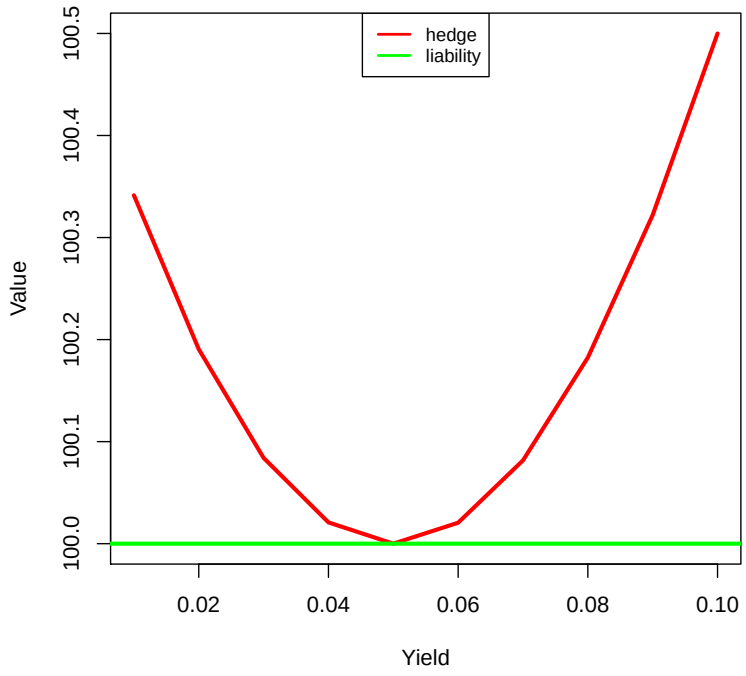



FIGURE 6.1: Scenario analysis of hedge value at investment horizon

```
> FutureValue <- function(coupon, n, yield, T) {
     100 * sum(sapply(1:n, function(i) ifelse(i < n, coupon/(1 +
         yield)^(i - T), (1 + coupon)/(1 + yield)^(n - T))))
 }
> r <- seq(0.01, 0.1, 0.01)
> V_A <- sapply(r, function(x) FutureValue(0.05, 3, x, 5))
> V_B <- sapply(r, function(x) FutureValue(0.06, 7, x, 5))
> Hedge <- q_A * V_A + q_B * V_B
```

Figure 6.1 plots the value of the investment portfolio under various yield scenarios. We observe that, under the restrictive conditions of this experiment, the hedge has always a greater value than the liability at horizon $T$. One should keep in mind, however, that this experiment is done under the assumption that yields for all maturities are identical, and that they all move up or down in parallel. More sophisticated methods address these limitations. The most popular method is to immunize separately various maturity segments. Another approach is to identify several yield

| Year | Liability |
|---|---|
| 1 | 100 |
| 2 | 200 |
| 3 | 150 |
| 4 | 400 |
| 5 | 300 |

Table 6.2: Liability cash flow stream

| Bond | Maturity | Coupon | Yield |
|---|---|---|---|
| 1 | 1 | .05 | .050 |
| 2 | 2 | .07 | .075 |
| 3 | 3 | .06 | .058 |
| 4 | 4 | .05 | .049 |
| 5 | 5 | .08 | .081 |

Table 6.3: Available bonds for dedication

curve risk factors, such as a parallel shift, a change in slope and a change in convexity, and immunize the liability against movements along each factor.
A completely different approach is to construct a portfolio that generates the necessary cash flow just in time, so as to minimize the need to re-invest cash that becomes available too early, or sell cash flows that occur too late. This strategy, called Dedication or cash-flow matching, is now described.

## 6.2 DEDICATION

Assume that, over the next 5 years, you must pay the amounts $L(t), t = 1,\ldots,5$ summarized in Table 6.2. In order to meet this liability, you can invest in a portfolio of 5 bonds described in table 6.3.
At every period, you can re-invest excess cash flow at a rate $r$, chosen conservatively, but cannot borrow. You want, of course, to construct the cheapest portfolio that will match the liability.
To formalize the strategy, we will use the following notation:

$q_i$ quantity of bond $i$, purchased at time $t = 0$

$C(t)$ cash balance at time $t$

$F_i(t)$ cash flow from 1 unit of bond $i$ at time $t$.

$P_i$ current price of bond $i$

The cash balance at each period is the compounded value of the cash balance from the previous period, augmented of the difference between the inflows and the outflows:

$$C(t) = (1+r)C(t-1) + \sum_i q_i F_i(t) - L(t)$$

The objective is to determine $q_i, i = 1, \ldots, 5$ and the initial cash balance $C(0)$, so that:

- The cost of the initial portfolio is minimized
- The portfolio is long assets ($q_i >= 0$)
- The cash balance is always positive

The investment strategy can be determined by solving the following linear program:

$$\begin{aligned} \min \quad & \sum_i q_i P_i && (6.1) \\ \text{s.t.} \quad & \\ & (1+r)C(t-1) + \sum_i q_i F_i(t) - C(t) = L(t) && (6.2) \\ & q_i >= 0, i = 1, \ldots, n \\ & C(t) >= 0, t = 1, \ldots, 5 \end{aligned}$$

There is no need to define an initial cash balance because there is already a one-year bond in the investment portfolio. It is convenient to express the problem in matrix notation:

$$\begin{aligned} \min \quad & P^T x && (6.3) \\ \text{s.t.} \quad & \\ & Ax = b \\ & x >= 0 \end{aligned}$$

where the column vector $x$ represents the variables of the problem:

$$x = (q_1, \ldots, q_5, c_1, \ldots, c_5)^T$$

and $P$ the vector of asset prices at time $t = 0$:

$$P = (P_1, \ldots, P_5, 0, 0, 0, 0, 0)^T$$

The cash flow definition equations in matrix form are defined with:

| | Quantity |
|---|---|
| B1 | 0.41 |
| B2 | 1.44 |
| B3 | 1.04 |
| B4 | 3.60 |
| B5 | 2.78 |

Table 6.4: Dedicated Portfolio, solution of problem 6.1-6.2

$$b = \begin{pmatrix} 100 \\ 200 \\ 150 \\ 400 \\ 300 \end{pmatrix}$$

$$A = \begin{pmatrix} 105 & 7 & 6 & 5 & 8 & -1 & 0 & 0 & 0 & 0 \\ 0 & 107 & 6 & 5 & 8 & 1+r & -1 & 0 & 0 & 0 \\ 0 & 0 & 106 & 5 & 8 & 0 & 1+r & -1 & 0 & 0 \\ 0 & 0 & 0 & 105 & 8 & 0 & 0 & 1+r & -1 & 0 \\ 0 & 0 & 0 & 0 & 108 & 0 & 0 & 0 & 1+r & -1 \end{pmatrix}$$

The program is set up as follows:

```
> r <- 0.01
> Amat = rbind(c(105, 7, 6, 5, 8, -1, 0, 0, 0, 0), c(0, 107, 6,
      5, 8, 1 + r, -1, 0, 0, 0), c(0, 0, 106, 5, 8, 0, 1 + r, -1,
      0, 0), c(0, 0, 0, 105, 8, 0, 0, 1 + r, -1, 0), c(0, 0, 0,
      0, 108, 0, 0, 0, 1 + r, -1))
```

The right hand side of 6.2:

```
> bvec = c(100, 200, 150, 400, 300)
> names(bvec) <- c("T1", "T2", "T3", "T4", "T5")
```

The vector $P$, the cost of assets at time $t = 0$:

```
> y <- 0.06
> cvec = c(SimpleBondPrice(0.05, 1, y), SimpleBondPrice(0.07, 2,
      y), SimpleBondPrice(0.06, 3, y), SimpleBondPrice(0.05, 4,
      y), SimpleBondPrice(0.08, 5, y), 0, 0, 0, 0, 0)
> names(cvec) <- c("B1", "B2", "B3", "B4", "B5", "C1", "C2", "C3",
      "C4", "C5")
```

The program is solved by

```
> lp.sol <- solveLP(cvec, bvec, Amat, const.dir = rep(">=", length(bvec)))
```

| | Period | Cash Balance |
|---|---|---|
| C1 | 1.00 | 0.00 |
| C2 | 2.00 | 108.70 |
| C3 | 3.00 | 0.00 |
| C4 | 4.00 | 0.00 |
| C5 | 5.00 | 0.00 |

Table 6.5: Bond dedication solution with 4 bonds. A cash balance in period 2 is needed to batch the liability.

The optimal portfolio composition is shown in Table 6.4.
Since we can have as many bonds as time period, and that the $A$ matrix is full row-rank, we have a perfect match between the generated cash flows and the liability stream, at a cost that is therefore exactly equal to the present value of the liability:

```
> df <- sapply(1:5, function(i) 1/(1 + y)^i)
> PV <- sum(df * bvec)
> print(paste("dedication cost:", round(lp.sol$opt, 2), "PV of liability:",
     round(PV, 2)))
[1] "dedication cost: 939.3 PV of liability: 939.3"
```

Any discrepancy would provide an opportunity for risk-less arbitrage.
Consider now the same problem, but with fewer bonds available to construct the dedicated portfolio (bond 3 is deleted from the set):

```
> Amat = rbind(c(105, 7, 5, 8, -1, 0, 0, 0, 0), c(0, 107, 5, 8,
     1 + r, -1, 0, 0, 0), c(0, 0, 5, 8, 0, 1 + r, -1, 0, 0), c(0,
     0, 105, 8, 0, 0, 1 + r, -1, 0), c(0, 0, 0, 108, 0, 0, 0,
     1 + r, -1))
> cvec = c(SimpleBondPrice(0.05, 1, y), SimpleBondPrice(0.07, 2,
     y), SimpleBondPrice(0.05, 4, y), SimpleBondPrice(0.08, 5,
     y), 0, 0, 0, 0, 0)
> names(cvec) <- c("B1", "B2", "B4", "B5", "C1", "C2", "C3", "C4",
     "C5")
> res2 <- solveLP(cvec, bvec, Amat, const.dir = rep(">=", length(bvec)))
```

The optimal portfolio is displayed in table 6.6. We no longer have a perfect match, and some cash must be reinvested from period to period, as shown in table 6.5: a cash balance is re-invested from period 2 to period 3.

```
> print(paste("dedication cost:", round(res2$opt, 2), "PV of liability:",
     round(PV, 2)))
[1] "dedication cost: 943.86 PV of liability: 939.3"
```

The dedication cost is naturally higher than the present value of the liability, since the solution involves reinvesting cash from the second to the third period, at a rate that is lower than the current yield.

| | Quantity |
|---|---|
| B1 | 0.40 |
| B2 | 2.51 |
| B4 | 3.60 |
| B5 | 2.78 |

Table 6.6: Bond dedication solution: optimal holdings with 4 bonds

Cash-flow matching is a popular strategy for managing short-term liabilities, because it minimizes transactions. It would not be economical to use such strategy for long term liabilities, of for complex cash flows. There is therefore, a natural division of labor between the two strategies that we have sketched.

# Part III

# Discreet Models for Derivatives

# CHAPTER 7

# RISK-NEUTRAL PRICING IN A BINOMIAL FRAMEWORK

In this chapter we use the binomial tree framework to introduce the key concepts of option valuation.

## 7.1 INTRODUCTION

Consider first a one-period model: An asset $S_t$ is worth \$40 at $t = 0$, and suppose that a month later, at $t = T$, it will be either \$45 or \$35. We are interested in buying a call option struck at $K = 40$, expiring at $T$. Interest rate is 1% per month. What is the fair price of this option?
The option payoff is

$$c_T = (S_T - K)^+ = \begin{cases} 5 & \text{if } S_T = 45 \\ 0 & \text{if } S_T = 35 \end{cases}$$

Now consider a portfolio made of one unit of stock and 2 call options:

$$\Phi = S - 2c$$

$$S_T - 2c_T = \begin{cases} 35 & \text{if } S_T = 45 \\ 35 & \text{if } S_T = 35 \end{cases}$$

This portfolio is worth today the same as a bank deposit that would provide \$35 in a month, or

$$\frac{35}{1+1\%} = 34.65 \tag{7.1}$$

In this simple one-period economy, the option is thus worth:

$$c_o = \frac{40 - 34.65}{2} = 2.695 \tag{7.2}$$

| State | Stock | Bond | Call |
|---|---|---|---|
| Today | $S_0$ | $B_0$ | $c_0$ |
| Up | $S_T^u = S_0 u$ | $(1+rT)B_0$ | $c_T^u = (S_T^u - K)^+$ |
| Down | $S_T^d = S_0 d$ | $(1+rT)B_0$ | $c_T^d = (S_T^d - K)^+$ |

Table 7.1: The two future states of a one-period binomial model

*The Binomial Model for Stocks*

Consider again the one-period binomial model of the previous section, and introduce notation to characterize the value of the three assets of interest, today and in the two future states, labeled "Up" and "Down". The notation is summarized in Table 7.1.
Construct a risk-free portfolio made of the option and the stock:

$$\Pi_0 = c_0 - \Delta S_0$$

To be riskless, one must have:

$$\Pi_T = (1+rT)\Pi_0$$

In particular, this is true in the two scenarios for $S_T$:

$$\begin{aligned} c_T^u - \Delta S_0 u &= (1+rT)(c_0 - \Delta S_0) \\ c_T^d - \Delta S_0 d &= (1+rT)(c_0 - \Delta S_0) \end{aligned}$$

Solve for $\Delta$:

$$\Delta = \frac{c_T^u - c_T^d}{S_0(u-d)}$$

Set $(1+rT) = \rho$. The option value at time $t = 0$ is:

$$\begin{aligned} c_0 &= \frac{1}{\rho}\Pi_T + \Delta S_0 \\ &= \frac{1}{\rho}\left(\frac{\rho - d}{u-d}c_T^u + \frac{u-\rho}{u-d}c_T^d\right) \end{aligned}$$

Assume $d < \rho < u$. and define:

$$\begin{aligned} q_u &= \frac{\rho - d}{u-d} \\ q_d &= \frac{u-\rho}{u-d} \end{aligned}$$

Rewrite:

$$c_0 = \frac{1}{\rho}\left(q_u c_T^u + q_d c_T^d\right) \tag{7.3}$$

One can observe that $0 < q_u, q_d < 1$ and that $q_u + q_d = 1$, and therefore interpret $q_u$ and $q_d$ as probabilities associated with the events $S_T = S_T^u$ and $S_T = S_T^d$. Let $Q$ be this probability measure. This leads us to write:

$$c_0 = \frac{1}{\rho} E^Q(c_T)$$

The option price is the discounted expected future payoff, under the probability $Q$.

## 7.2 PRICING WITH ARROW-DEBREU SECURITIES

An alternative derivation of the same result can be obtained with Arrow-Debreu securities. Let's first compute the price $a_1$ and $a_2$ of the two Arrow-Debreu securities in this economy. The price of the option will then be, by definition:

$$c_0 = a_1 c_T^u + a_2 c_T^d$$

where $a_1$ and $a_2$ are the prices of the Arrow-Debreu securities for the up and down scenarios.
Let's now determine the prices of these Arrrow-Debreu securities. To do so, we construct a portfolio made of $x$ units of stock and $y$ units of bond, that has the same payoff as an Arrrow-Debreu security. Setting $B_0 = 1$, the replicating portfolio for the first Arrow-Debreu security is obtained by solving the system:

$$\begin{pmatrix} S_0 u & \rho \\ S_0 d & \rho \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Which yields:

$$x = \frac{1}{S_0(u-d)}, \quad y = -\frac{d}{\rho(u-d)}$$

The price of the first Arrow-Debreu security is thus:

$$\begin{aligned} a_1 &= x S_0 + y B_0 \\ &= \frac{1}{\rho}\frac{\rho - d}{u - d} \end{aligned}$$

Similarly, the second Arrow-Debreu security is found to be worth:

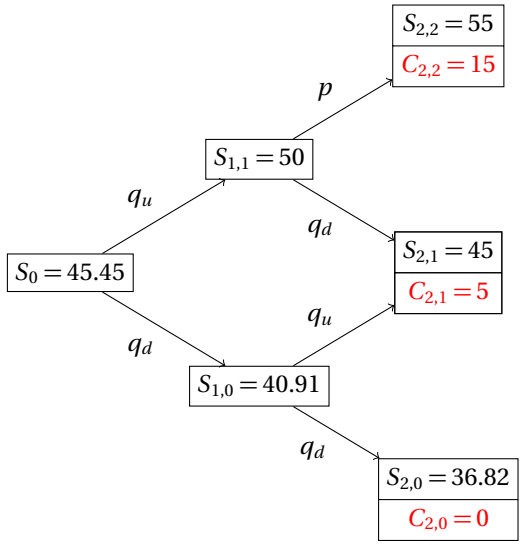



FIGURE 7.1: A 2-period binomial tree, showing the exercise value of a call option struck at 40.

$$a_2 = \frac{1}{\rho}\frac{\rho - u}{u - d}$$

and we obtain therefore the same option price as in (7.3)

## 7.3 MULTI-PERIOD BINOMIAL MODEL

The logic of the previous sections can be extented to multiple periods. When dealing with multiple periods, it is important in practice to construct recombining trees, i.e. trees where an up move followed by a down move results in the same state than a down move followed by a up move. $N$ steps in a recombining tree generate $N+1$ states, but $2^N$ states in a non-recombining binomial tree.

The calculation process in a multi-period tree is best explained through an example. Consider a stock priced at 45.45€. At each period, the stock may appreciate or depreciate by 10%. The riskless rate is 5%, and we want to price a call option struck at 40€. The two-period tree is represented in figure 7.1, with stock price in black and the option exercise value in red.

The risk-neutral probabilities are identical at all nodes:

$$\begin{aligned} q_u &= \frac{\rho - d}{u - d} = \frac{1.05 - .9}{1.1 - .9} = .75 \\ q_d &= .25 \end{aligned}$$

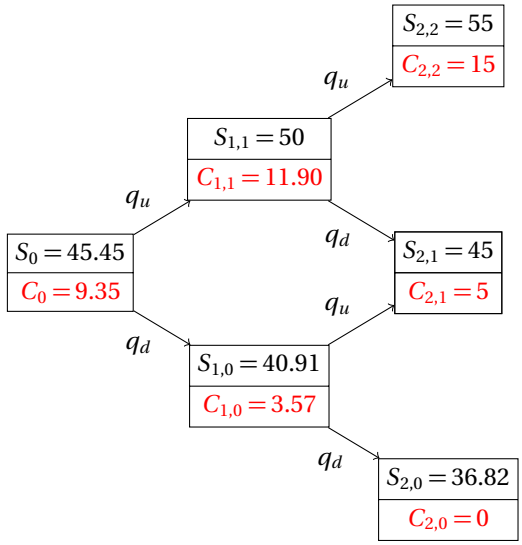



FIGURE 7.2: 2-step binomial tree with price of a call option struck at 40

Using these values, the option value one step prior to expiry can be computed using (7.3):

$$\begin{aligned} C_{1,1} &= \frac{1}{\rho}(q_u C_{2,2} + q_d C_{2,1}) = 11.90 \\ C_{1,0} &= \frac{1}{\rho}(q_u C_{2,1} + q_d C_{2,0}) = 3.57 \end{aligned}$$

The reasoning that led to (7.3) applies however to any node, and in particular to node (0,0). The option price at that node is thus:

$$C_{0,0} = \frac{1}{\rho}(q_u C_{1,1} + q_d C_{1,0}) = 9.35$$

The process is pictured in Figure 7.2.

## 7.4 AMERICAN EXERCISE

American exercise refer to the right to exercise the option at any time before expiry. Clearly, an option with American exercise is worth more than the comparable option with European exercise. To price an American option, we introduce the notion of continuation value. The continuation value $V_t^i$ at node $i$ and time $t$ is the option value, assuming that it will be exercised after time $t$. At each step, one determines the optimal decision

by comparing the value of an immediate exercise to the continuation value. The option value is therefore, for a call:

$$\begin{aligned} C_t^i &= \max(S_t^i - K, V_t^i) \\ &= \max(S_t^i - K, \frac{1}{\rho}(q_u C_{t+1}^{i+1} + q_d C_{t+1}^i)) \end{aligned}$$

**Example** Starting from the data of the previous section, we now assume that the stock pays a dividend of 3 € in period 2. The modified tree is represented in figure 7.3.
We price an American call struck at 40€ in this tree. Exercise values in period 2 are computed as before, giving the following values:

$$\begin{aligned} C_{2,2} &= 12 \\ C_{2,1} &= 0 \\ C_{2,0} &= 0 \end{aligned}$$

At period 1, the option holder determines the optimal policy: exercise immediately or hold the option until period 2. The resulting values are:

$$\begin{aligned} C_{1,1} &= \max((50-40), \frac{1}{\rho}(q_u 10 + q_d 2)) = 10 \\ C_{1,0} &= \max((40.91-40), \frac{1}{\rho}(q_u 2 + q_d 0)) = 1.42 \end{aligned}$$

The option value today is finally determined to be:

$$\begin{aligned} C_{0,0} &= \max(5.45, \frac{1}{\rho}(q_u 10 + q_d 1.42)) \\ &= 7.48 \end{aligned}$$

Under the same conditions, a European option is worth $C_{0,0} = 6.79$. The difference comes from the early exercise of the American option in node (1,1).

## 7.5 CALIBRATION OF THE BINOMIAL MODEL

With interest rate assumed to be known, option prices are determined by the terms $u$ and $d$ that describe the binomial branching process. How should $u$ and $d$ be chosen?

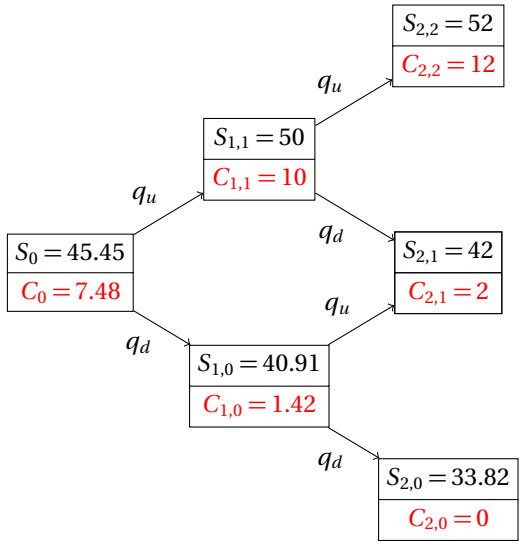



FIGURE 7.3: 2-step binomial tree

The time interval $[0,T]$ is divided into $N$ equal steps $[t_j, t_{j+1}]$ of length $\Delta t$. Assume that the process for the underlying asset $S_t$ is such that $V_t = \ln(S_t/S_{t-\Delta t})$ are iid random variables with mean $\mu\Delta t$ and variance $\sigma^2\Delta t$.

$$S_{t_j} = S_{t_{j-1}} e^{V_j}$$

Let's determine $u$ and $d$ by matching the mean and variance of $V_t$ and of the binomial process:

$$\begin{aligned} E(V_j) &= p\ln u + (1-p)\ln d \\ &= \mu\Delta t \\ V(V_j) &= E(V_j^2) - E(V_j)^2 \\ &= \sigma^2\Delta t \end{aligned}$$

Which forms a system of 2 equations and three unknown. Without loss of generality, set $p = 1/2$, to obtain:

$$\begin{aligned} \frac{1}{2}(\ln u + \ln d) &= \mu\Delta t \\ \frac{1}{4}(\ln u + \ln d)^2 &= \sigma^2\Delta t \end{aligned}$$

The solution is:

$$\begin{aligned} u &= e^{\mu\Delta t+\sigma\sqrt{\Delta t}} \\ d &= e^{\mu\Delta t-\sigma\sqrt{\Delta t}} \end{aligned} \tag{7.4}$$

The corresponding value of the risk-neutral up probability, $q_u$ is

$$q_u = \frac{e^{r\Delta t} - e^{\mu\Delta t-\sigma\sqrt{\Delta t}}}{e^{\mu\Delta t+\sigma\sqrt{\Delta t}} - e^{\mu\Delta t-\sigma\sqrt{\Delta t}}} \tag{7.5}$$

This is the single-period risk-neutral probability, not the objective probability $p$. We next compute the limit of $q_u$ as $\Delta t \to 0$.
We use the following first order approximation:

$$\begin{aligned} e^{\mu\Delta t+\sigma\sqrt{\Delta t}} &= (1+\mu\Delta t+\sigma\sqrt{\Delta t}+\frac{1}{2}(\mu\Delta t+\sigma\sqrt{\Delta t})^2+\ldots \\ &= 1+\mu\Delta t+\sigma\sqrt{\Delta t}+\frac{1}{2}\sigma^2\Delta t+O(\Delta t^{3/2}) \end{aligned}$$

and similarly,

$$e^{\mu\Delta t-\sigma\sqrt{\Delta t}} = 1+\mu\Delta t-\sigma\sqrt{\Delta t}+\frac{1}{2}\sigma^2\Delta t+O(\Delta t^{3/2})$$

Combining these approximations with (7.5) yields,

$$\begin{aligned} q_u &= \frac{\sigma\sqrt{\Delta t}+(r-\frac{1}{2}\sigma^2-\mu)\Delta t+O(\Delta t^{3/2})}{2\sigma\sqrt{\Delta t}+O(\Delta t^{3/2})} \\ &= \frac{1}{2}+\frac{r-\frac{1}{2}\sigma^2-\mu}{2\sigma}\sqrt{\Delta t}+O(\Delta t) \end{aligned} \tag{7.6}$$

Let's now use these expressions for $q_u$ to compute the mean and variance of $V_j$.

$$E(V_j) = q_u \ln u + (1-q_u)\ln d$$

Use (7.4) to get:

$$E(V_j) = q_u(2\sigma\sqrt{\Delta t})+\mu\Delta t-\sigma\sqrt{\Delta t}$$

Substitute $q_u$ by its value (7.5) to get:

$$E(V_j) = (r-\frac{1}{2}\sigma^2)\Delta t+O(\Delta t^{3/2})$$

Similarly,

$$Var(V_j) = \sigma^2\Delta t+O(\Delta t^{3/2})$$

The remarkable point of this result is that $\mu$ no longer appears.
Extending the reasoning of Section 7.1 to multiple periods, we write that under the risk neutral probability $q_u, q_d$, the option value is the discounted expected value of the payoff:

$$\begin{aligned} P(S_0) &= e^{-rT} E^Q(f(S_T)) \\ &= e^{-rT} E^Q(f(S_0 e^{\sum_{i=1}^N V_i})) \end{aligned} \tag{7.7}$$

where $f(S_T)$ is the payoff at expiry. We next need to compute the limit:

$$\lim_{N\to\infty} \sum_{i=1}^{N} V_i$$

The variables $V_i$ are a function of $N$, thus the Central Limit Theorem cannot be used as such. However, we can invoke Lindeberg's condition to obtain the same result:

**Theorem 7.5.1.** (Lindeberg's Condition) *Let $X_k, k \in N$ be independent variables with $E(X_k) = \mu_k$ and $V(X_k) = \sigma_k^2$. Let $s_n^2 = \sum_i^n \sigma_i^2$. If the variables satisfy Lindeberg's condition, then*

$$Z_n = \frac{\sum_{k=1}^n (X_k - \mu_k)}{s_n} \to N(0,1)$$

To simplify notation, let $E^Q(V_i) = a$, $Var^Q(V_i) = b$, Lindeberg's condition yields:

$$\frac{\sum_i^N V_i - Na}{b\sqrt{N}} \to N(0,1)$$

$$\frac{\sum_i^N V_i - Na}{b\sqrt{N}} = \frac{\sum_i^N V_i - (r - \frac{1}{2}\sigma^2)T + O(N^{-1/2})}{\sigma\sqrt{T} + O(N^{-1/2})}$$

$$\frac{\sum_i^N V_i - (r - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T})} \to N(0,1)$$

Thus,

$$\sum_i^N V_i \to N\left((r - \frac{1}{2}\sigma^2)T, \sigma^2 T\right)$$

Finally, as $N \to \infty$, (7.8) becomes:

$$P(S_0) = \frac{e^{-rT}}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(S_0 e^{(r-\frac{1}{2}\sigma^2)T + \sigma\sqrt{T}} e^{-\frac{1}{2}u^2} du$$

which is the Black-Scholes valuation formula, as derived by Cox, Ross & Rubinstein (1979). Again, the significance of this result is that $\mu$ does not appear in the formula.

*Tree Geometry*

We now use the result from the previous section to determine the geometry of the tree and the risk-neutral transition probabilities, consistent with the parameters of the diffusion process.
Recall from the previous section that $u$ and $d$ are defined by:

$$\begin{aligned} u &= e^{\mu\Delta t+\sigma\sqrt{\Delta t}} \\ d &= e^{\mu\Delta t-\sigma\sqrt{\Delta t}} \end{aligned}$$

Ultimately, $\mu$ does not appear in the valuation formula, it can thus be set to an arbitrary value without loss of generality.
In the original CRR model, $\mu = 0$, which leads to:

$$\begin{aligned} u &= e^{\sigma\sqrt{t}} \\ d &= e^{-\sigma\sqrt{t}} \\ q_u &= \frac{e^{rt}-e^{-\sigma\sqrt{t}}}{e^{\sigma\sqrt{t}}-e^{-\sigma\sqrt{t}}} \end{aligned}$$

There are many other possibilities: a popular choice introduced by R. Jarrow (1993) is to set $\mu$ so that transition probabilities are equal to $\frac{1}{2}$. Using (7.6), this involves setting $\mu = r - \frac{1}{2}\sigma^2$, and the branching process is then:

$$\begin{aligned} u &= e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\sqrt{\Delta t}} \\ d &= e^{(r-\frac{1}{2}\sigma^2)\Delta t-\sigma\sqrt{\Delta t}} \\ q_u &= \frac{1}{2} \end{aligned}$$

There are many other possible choices, but no significant differences in the convergence rate to the Black-Scholes option value. Most of the models, however, suffer from a form of instability which is now described.

## 7.6 STABILITY OF THE BINOMIAL MODEL

*Numerical Test*

We would like to verify the convergence of the Cox-Ross model to the Black-Scholes price as the number of steps $N$ increases. This can be investigated with the following script:

```
> Strike <- 100
> Spot <- 100
> T1 <- 1/4
```

```
> r <- 0.05
> b <- 0.05
> sigma <- 0.3
> NN <- seq(20, 100, 1)
> nb <- length(NN)
> res <- matrix(nrow = nb, ncol = 2)
> bs <- rep(0, 2)
> bs[1] <- GBSOption(TypeFlag = "c", S = Spot, X = Strike, Time = T1,
      r = r, b = b, sigma = sigma)@price
> bs[2] <- GBSOption(TypeFlag = "c", S = Spot, X = Strike + 10,
      Time = T1, r = r, b = b, sigma = sigma)@price
> res[, 1] <- sapply(NN, function(n) CRRBinomialTreeOption(TypeFlag = "ce",
      S = Spot, X = Strike, Time = T1, r = r, b = b, sigma = sigma,
      n)@price)
> res[, 2] <- sapply(NN, function(n) CRRBinomialTreeOption(TypeFlag = "ce",
      S = Spot, X = Strike + 10, Time = T1, r = r, b = b, sigma = sigma,
      n)@price)
```

A plot of the prices as a function of the number of steps (Figure 7.4) shows an oscillating pattern:

```
> par(mfrow = c(1, 2))
> plot(NN, res[, 1], type = "l", main = "ATM", xlab = "Number of steps",
      ylab = "price", col = "blue", ylim = c(min(res[, 1]), max(res[,
          1])))
> abline(h = bs[1], lwd = 2, col = "red")
> plot(NN, res[, 2], type = "l", main = "OTM", xlab = "Number of steps",
      ylab = "price", col = "blue", ylim = c(min(res[, 2]), max(res[,
          2])))
> abline(h = bs[2], lwd = 2, col = "red")
> par(mfrow = c(1, 1))
```

Other binomial algorithms, such as Tian's, exhibit a similar pattern, as evidenced in Figure 7.5. The horizontal line marks the Black-Scholes price.

```
> res[, 1] <- sapply(NN, function(n) TIANBinomialTreeOption(TypeFlag = "ce",
      S = Spot, X = Strike, Time = T1, r = r, b = b, sigma = sigma,
      n)@price)
> res[, 2] <- sapply(NN, function(n) TIANBinomialTreeOption(TypeFlag = "ce",
      S = Spot, X = Strike + 10, Time = T1, r = r, b = b, sigma = sigma,
      n)@price)
```

This sawtooth pattern is due to the position of the strike relative to the sequence of nodes at expiry. We consider next several strategies for smoothing these oscillations.

*Speeding up the Convergence*

Since the oscillations are caused by the variations in the relative position of the strike with respect to nodes at expiry, a natural strategy, introduced by Leisen & Reimer (1996), is to construct the tree such that the strike

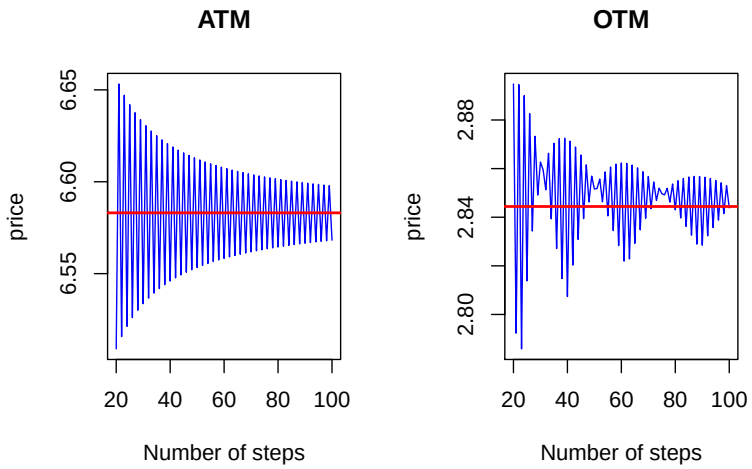



FIGURE 7.4: Comparison between the CRR binomial model and the Black-Scholes price as a function of the number of time steps in CRR. Left panel: an at-the-money option. Right panel: an out-of-the-money option.

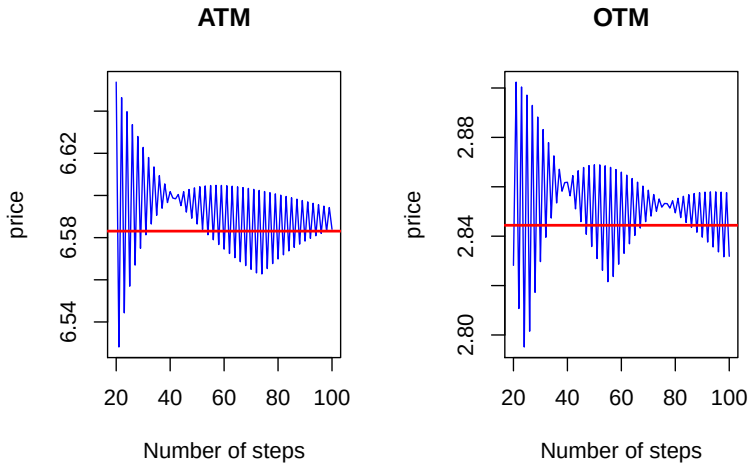



FIGURE 7.5: Convergence of European Call in Tian's Binomial Model

coincides with a node. This is achieved by setting

$$\mu = \frac{1}{T}\log\left(\frac{K}{S_0}\right)$$

The resulting tree is centered on $K$ in log space. The pricing method is implemented as follows:

```
> CRRWithDrift <- function(TypeFlag = c("ce", "pe", "ca", "pa"),
     S, X, Time, r, mu, sigma, n) {
     TypeFlag = TypeFlag[1]
     z = NA
     if (TypeFlag == "ce" || TypeFlag == "ca")
         z = +1
     if (TypeFlag == "pe" || TypeFlag == "pa")
         z = -1
     if (is.na(z))
         stop("TypeFlag misspecified: ce|ca|pe|pa")
     dt = Time/n
     u = exp(mu * dt + sigma * sqrt(dt))
     d = exp(mu * dt - sigma * sqrt(dt))
     p = (exp(r * dt) - d)/(u - d)
     Df = exp(-r * dt)
     ST <- S * (d^(n - 1)) * cumprod(c(1, rep((u/d), n - 1)))
     BSTypeFlag <- substr(TypeFlag, 1, 1)
     OptionValue <- GBSOption(BSTypeFlag, ST, X, dt, r, b, sigma)@price
     if (TypeFlag == "ce" || TypeFlag == "pe") {
         for (j in seq(from = n - 2, to = 0, by = -1)) OptionValue <- (p *
             OptionValue[2:(j + 2)] + (1 - p) * OptionValue[1:(j +
             1)]) * Df
     }
     if (TypeFlag == "ca" || TypeFlag == "pa") {
         for (j in seq(from = n - 2, to = 0, by = -1)) {
             ContValue <- (p * OptionValue[2:(j + 2)] + (1 - p) *
                 OptionValue[1:(j + 1)]) * Df
             ST <- S * (d^j) * cumprod(c(1, rep((u/d), j)))
             OptionValue <- sapply(1:(j + 1), function(i) max(ST[i] -
                 X, ContValue[i]))
         }
     }
     OptionValue[1]
 }
```

Convergence of the model as $N$ increases is significantly improved, as evidenced by the graphs in Figure 7.6.
In the context of an American option, note that if the option has not been exercised at step $N-1$, the option is now European, and can be priced at these nodes with the Black-Scholes model, rather than with the backward recursion from step $N$ (the expiry date). This simple modification smooths the option value at step $N-1$ and cancels the oscillations, as illustrated in figure 7.7, but at the price of a substantial increase in computation time.

```
> CRRWithBS <- function(TypeFlag = c("ce", "pe", "ca", "pa"), S,
```

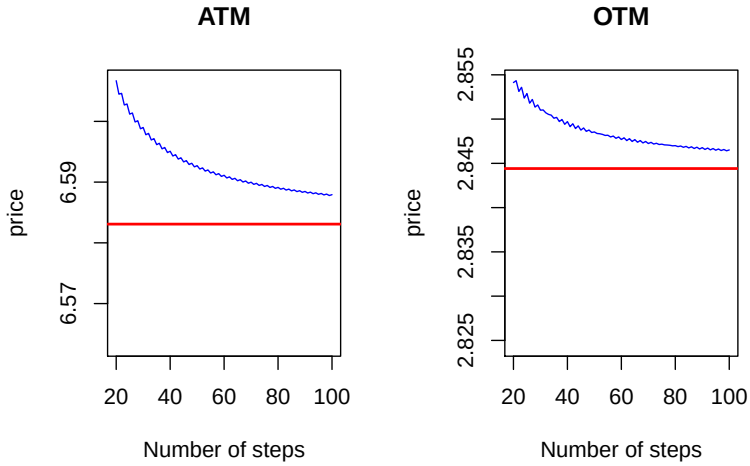



FIGURE 7.6: Convergence of CRR with Drift to Black-Scholes as a function of the number of time steps in CRR

```
X, Time, r, b, sigma, n) {
TypeFlag = TypeFlag[1]
z = NA
if (TypeFlag == "ce" || TypeFlag == "ca")
    z = +1
if (TypeFlag == "pe" || TypeFlag == "pa")
    z = -1
if (is.na(z))
    stop("TypeFlag misspecified: ce|ca|pe|pa")
dt = Time/n
u = exp(sigma * sqrt(dt))
d = 1/u
p = (exp(b * dt) - d)/(u - d)
Df = exp(-r * dt)
ST <- S * (d^(n - 1)) * cumprod(c(1, rep((u/d), n - 1)))
BSTypeFlag <- substr(TypeFlag, 1, 1)
OptionValue <- GBSOption(BSTypeFlag, ST, X, dt, r, b, sigma)@price
if (TypeFlag == "ce" || TypeFlag == "pe") {
    for (j in seq(from = n - 2, to = 0, by = -1)) OptionValue <- (p *
        OptionValue[2:(j + 2)] + (1 - p) * OptionValue[1:(j +
        1)]) * Df
}
if (TypeFlag == "ca" || TypeFlag == "pa") {
    for (j in seq(from = n - 2, to = 0, by = -1)) {
        ContValue <- (p * OptionValue[2:(j + 2)] + (1 - p) *
            OptionValue[1:(j + 1)]) * Df
        ST <- S * (d^j) * cumprod(c(1, rep((u/d), j)))
        OptionValue <- sapply(1:(j + 1), function(i) max(ST[i] -
            X, ContValue[i]))
```

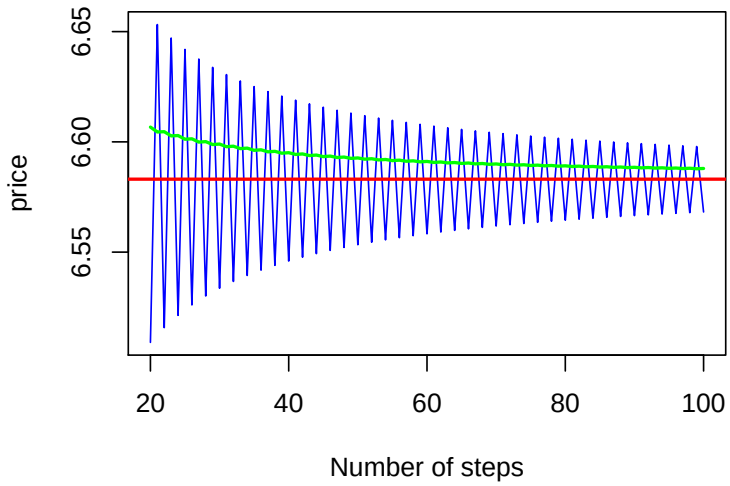



FIGURE 7.7: Accelerated Convergence of the Binomial Model

```
        }
    }
    OptionValue[1]
}
```

Joshi (2007) has conducted an extensive survey of binomial models with improved convergence properties.

## 7.7 TRINOMIAL MODELS

A natural extension of the binomial model is a trinomial model, that is, a model with three possible future states at each time step and current state.

The stock price at time $t$ is $S_t$. From $t$ to $t+\Delta t$, the stock may move up to $S_u$ with probability $p_u$, down to $S_d$ with probability $p_d$, or move to a middle state $S_m$ with probability $1-p_u-p_d$. The probabilities and future states must satisfy the following constraints:

1. The expected value of the stock at $t+\Delta t$ must be the forward price:

$$p_u S_u + p_d S_d + (1-p_u-p_d)S_m = F = Se^{(r-\delta)\Delta t}$$

2. Variance:

$$p_u(S_u - F)^2 + p_d(S_d - F)^2 + (1 - p_u - p_d)(S_m - F)^2 = S^2\sigma^2\Delta t$$

The first method for constructing trinomial trees is simply to combine two steps of any binomial tree.

*Aggregation of binomial trees*

Recall that a CRR binomial tree is defined by:

$$\begin{aligned} u &= e^{\sigma\sqrt{\Delta t}} \\ d &= e^{-\sigma\sqrt{\Delta t}} \\ p &= \frac{e^{rt} - e^{-\sigma\sqrt{\Delta t}}}{e^{\sigma\sqrt{\Delta t}} - e^{-\sigma\sqrt{\Delta t}}} \end{aligned}$$

Combining two steps at a time, we obtain a trinomial tree with

$$\begin{aligned} S_u &= Se^{\sigma\sqrt{2\Delta t}} \\ S_m &= S \\ S_d &= Se^{-\sigma\sqrt{2\Delta t}} \\ p_u &= \left(\frac{e^{r\Delta t} - e^{-\sigma\sqrt{\Delta t}}}{e^{\sigma\sqrt{\Delta t}} - e^{-\sigma\sqrt{\Delta t}}}\right)^2 \\ p_d &= (1 - \sqrt{p_u})^2 \end{aligned}$$

To every binomial tree corresponds a trinomial tree, obtained by aggregating two steps.

# Part IV

# Derivatives in the Black-Scholes Framework

# CHAPTER 8

# PRICE AND GREEKS IN THE BLACK-SCHOLES MODEL

In this chapter, we consider the price and risk indicators of several types of options. We are particularly interested in the higher order risk indicators, such as the Gamma. We will see later that this indicator has a major impact on our ability to dynamically hedge such instruments.

## 8.1 SOLVING THE PRICING EQUATION

A fairly rigorous derivation of the Black-Scholes equation can be obtained without using the tools of stochastic calculus. Recall from the previous section that under the risk-neutral probability, the discounted value is a martingale:

$$S_0 = e^{-rT} E^Q(S_T)$$

where $S_T$ is a log-normal variable that follows the process:

$$S_T = S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}Y}$$

with $Y$ $N(0,1)$. The value of any derivative is the discounted expected payoff. For a call option, we have:

$$\begin{aligned} C_{\text{BS}}(S,0) &= e^{-rT} E^Q[\left(S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}Y} - K\right)^+] \\ &= \frac{e^{-rT}}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left(S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}Y} - K\right)^+ e^{-x^2/2} dx \end{aligned}$$

Now compute the bounds of integration:

$$\begin{aligned} & S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}x} \geq K \\ <=> \quad & e^{\sigma\sqrt{T}x} \geq \frac{K}{S_0} e^{-(r-\frac{\sigma^2}{2})T} \\ <=> \quad & x \geq \frac{1}{\sigma\sqrt{T}} \left( \ln(\frac{K}{S_0}) - (r-\frac{\sigma^2}{2})T \right) \end{aligned}$$

Let

$$T_1 = \frac{1}{\sigma\sqrt{T}} \left( \ln(\frac{K}{S_0}) - (r-\frac{\sigma^2}{2})T \right)$$

$$\begin{aligned} c(S,0) &= \frac{e^{-rT}}{\sqrt{2\pi}} \int_{T_1}^{\infty} \left( S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}x} - K \right) e^{-x^2/2} dx \\ &= \frac{e^{-rT}}{\sqrt{2\pi}} \int_{T_1}^{\infty} S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}x} e^{-x^2/2} dx - K \frac{e^{-rT}}{\sqrt{2\pi}} \int_{T_1}^{\infty} e^{-x^2/2} dx \\ &= A - B \end{aligned}$$

$$B = K e^{-rT} (1 - N(T_1))$$

Apply the change of variable $y = x - \sigma\sqrt{T}$ to get

$$\begin{aligned} A &= \frac{e^{-rT}}{\sqrt{2\pi}} \int_{T_1-\sigma\sqrt{T}}^{\infty} S_0 e^{-y^2/2} dy \\ &= S_0 (1 - N(T_1 - \sigma\sqrt{T})) \end{aligned}$$

where $N(x)$ is the cumulative normal distribution. Using the identity $N(x) + N(-x) = 1$, one gets the Black-Scholes formula in its standard form.

$$C_{\mathrm{BS}}(S,0) = S_0 N(d_1) - K e^{-rT} N(d_2) \tag{8.1}$$

with

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

The value of the put can be obtained from the call-put parity relationships. In the following discussion about the management of market risk of option portfolios, we use three risk indicators, also known as greeks:

- Delta, which is the derivative of price with respect to the price of the underlying asset,
$$\Delta = \frac{\partial C}{\partial S}$$
- Gamma, which is the second derivative of price with respect to spot, and
$$\Gamma = \frac{\partial^2 C}{\partial S^2}$$
- Vega, which is the derivative of price with respect to the volatility.
$$\nu = \frac{\partial C}{\partial \sigma}$$

See Hull (1997) for a detailed discussion of these risk indicators.

## 8.2 AN OPTION DASHBOARD

We have constructed a "dashboard" to illustrate the properties of various types of European options. This display is inspired by a similar layout developped for the RQuantLib project (Eddelbuettel, 2009).
The `fInstrument` class provides a generic way of writing such function.
The dashboard has four panels, which display the price, delta, gamma and vega of the option under consideration. Each panel has a family of curves varying in color from yellow to green to blue. This family of curves pictures the evolution of the price and of the greeks for varying levels of volatility. The yellow curves corresponding to a high volatility conditions, and at the other end, the blue line corresponding to low volatility conditions.

```
> OptionDashBoard <- function(inst, base.env, dtCalc, und.seq,
    vol.seq) {
    sce <- t(as.matrix(und.seq))
    underlying <- inst@params$underlying
    setData(base.env, underlying, "Price", dtCalc, sce)
    p <- matrix(nrow = length(und.seq), ncol = length(vol.seq))
    d <- matrix(nrow = length(und.seq), ncol = length(vol.seq))
    g <- matrix(nrow = length(und.seq), ncol = length(vol.seq))
    v <- matrix(nrow = length(und.seq), ncol = length(vol.seq))
    for (i in seq_along(vol.seq)) {
        setData(base.env, underlying, "ATMVol", dtCalc, vol.seq[i])
        p[, i] <- getValue(inst, "Price", dtCalc, base.env)
        d[, i] <- getValue(inst, "Delta", dtCalc, base.env)
        g[, i] <- getValue(inst, "Gamma", dtCalc, base.env)
        v[, i] <- getValue(inst, "Vega", dtCalc, base.env)
    }
    old.par <- par(no.readonly = TRUE)
    par(mfrow = c(2, 2), oma = c(5, 0, 0, 0), mar = c(2, 2, 2,
        1))
    plot(und.seq, p[, 1], type = "n", main = "Price", xlab = "",
        ylab = "")
```

```
    topocol <- topo.colors(length(vol.seq))
    for (i in 2:length(vol.seq)) lines(und.seq, p[, i], col = topocol[i])
    plot(und.seq, d[, 1], type = "n", main = "Delta", xlab = "",
        ylab = "")
    for (i in 2:length(vol.seq)) lines(und.seq, d[, i], col = topocol[i])
    plot(und.seq, g[, 1], type = "n", main = "Gamma", xlab = "",
        ylab = "")
    for (i in 2:length(vol.seq)) lines(und.seq, g[, i], col = topocol[i])
    plot(und.seq, v[, 1], type = "n", main = "Vega", xlab = "",
        ylab = "")
    for (i in 2:length(vol.seq)) lines(und.seq, v[, i], col = topocol[i])
    mtext(text = paste(inst@desc, "\nrate: 0.03", "Underlying from",
        und.seq[1], "to", und.seq[length(und.seq)], "Volatility  from",
        vol.seq[1], "to", vol.seq[length(vol.seq)]), side = 1,
        font = 1, outer = TRUE, line = 3)
    par(old.par)
}
```

*Vanilla Option*

European calls and puts are called "vanilla" in the banking industry, presumably due to their simple structures. The payoff of a vanilla option (call) is:

$$(S_T - K)^+$$

and its value (call) is given by:

$$e^{-rT}(F_T N(d_1) - K N(d_2))$$

where $F_T$ is the forward value of the underlying asset at $T$.
For at-the-money options, a simple approximation to the Black-Scholes price is:

$$V = e^{-rT} F_T \frac{\sigma\sqrt{T}}{\sqrt{2\pi}}$$

The formula above makes use of the approximation:

$$N(x) \approx \frac{1}{2} + \frac{x}{\sqrt{2\pi}} + O(x^2)$$

The evolution of price and greeks as a function of the underlying asset and volatility is shown in figure 8.1. The graph is obtained with the following script:

```
> dtExpiry <- myDate("01jan2011")
> underlying <- "IBM"
> Strike <- 100
> K <- 1
> b <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
     strike = Strike, dtExpiry = dtExpiry, underlying = underlying,
     discountRef = "USD.LIBOR", trace = F))
```

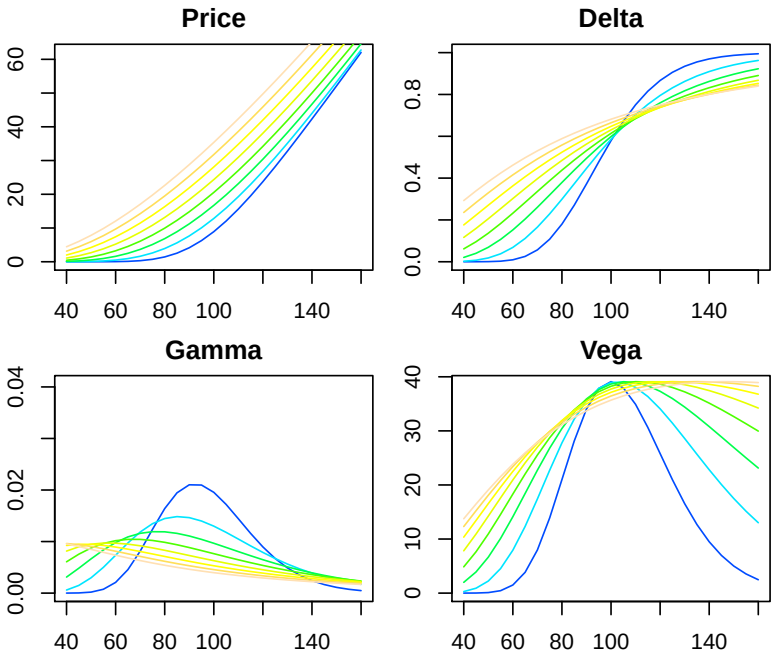



FIGURE 8.1: Price and greeks on a European call

```
> und.seq <- seq(40, 160, by = 5)
> vol.seq <- seq(0.1, 0.9, by = 0.1)
> base.env <- DataProvider()
> dtCalc <- myDate("01jan2010")
> setData(base.env, underlying, "DivYield", dtCalc, 0.02)
> setData(base.env, "USD.LIBOR", "Yield", dtCalc, 0.02)
> OptionDashBoard(b, base.env, dtCalc, und.seq, vol.seq)
```

Several observations are worth mentioning:

- In high volatility conditions, the delta is almost linear, but takes on more pronounced a "S" shape as volatility decrease.
- The gamma is always positive, reflecting the uniformly convex shape of the premium function.
- The vega is also always positive: an increase in volatility always results in an increase in premium.

We will see shortly that these last two features have important implications for risk management.

*Binary Option*

A binary option pays a fixed amount if the underlying asset is in the money at expiry.
The payoff is:

$$1_{S_T>K}$$

Recall that $N(d_2)$ is the risk-neutral probability that $S_T > K$ at expiry. The value of a binary call is therefore:

$$e^{-rT}N(d_2)$$

Alternatively, observe that a binary option struck at $K$ can be replicated by a very tight call spread:

$$C_B(K) = \lim_{h\to 0} \frac{C_V(K-h) - C_V(K-h)}{h}$$

where $C_B(K)$ is the price of a binary call struk at $K$and $C_V(K)$ the price of a vanilla call. The value of a binary call is the negative of the derivative of a vanilla call price with respect to strike.

$$\begin{aligned} C_B(K) &= -\frac{\partial C_V(K)}{\partial K} \\ &= e^{-rT}N(d_2) \end{aligned}$$

Recall that the derivative of a vanilla call price with respect to spot is $N(d_1)$, therefore, the shape of the binary price function is similar to the shape of the delta of a European option, and the shape of the delta of a binary call is similar to the shape of the gamma of a vanilla option.
As a consequence, the gamma of a binary option changes sign, and so does the vega. Options with gamma that changes sign are costly to hedge, as simulations will demonstrate in section.
The dashboard in Figure 8.2 is generated with the following script:

```
> dtExpiry <- myDate("01jan2011")
> underlying <- "IBM"
> Strike <- 100
> K <- 1
> bi <- fInstrumentFactory("binary", quantity = 1, params = list(cp = "c",
     strike = Strike, dtExpiry = dtExpiry, underlying = underlying,
     discountRef = "USD.LIBOR", trace = F))
> und.seq <- seq(40, 160, by = 5)
> vol.seq <- seq(0.1, 0.9, by = 0.1)
> base.env <- DataProvider()
> dtCalc <- myDate("01jan2010")
```

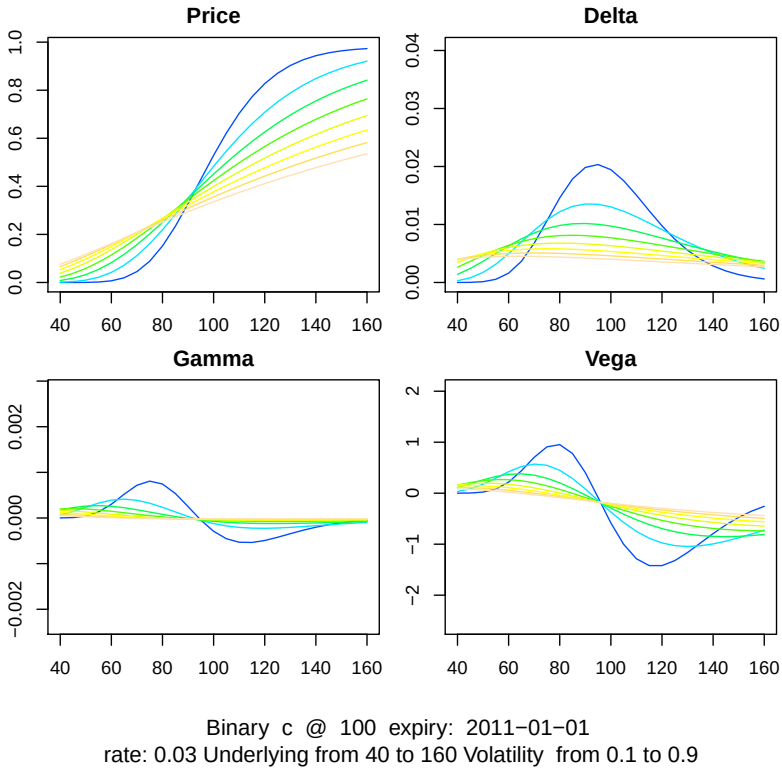



FIGURE 8.2: European Binary Call Strike=100

```
> setData(base.env, underlying, "DivYield", dtCalc, 0.02)
> setData(base.env, "USD.LIBOR", "Yield", dtCalc, 0.02)
> OptionDashBoard(bi, base.env, dtCalc, und.seq, vol.seq)
```

*Barrier Option*

There is a large variety of barrier options. Haug [Haug (2006)] provides a long list of pricing formulae for standard and complex barrier payoffs. As an illustration we consider here the "up and out" call, which is a regular European call that is canceled if the underlying asset breaches an upper barrier before expiry.
Payoff:

$$(S_T - K)^+ 1_{\max_t S_t < B}$$

This instruments introduces a new twist in the option landscape, since this call option can have a negative delta. As for binary options, the gamma

can switch sign, and finally, observe the magnitude of the vega: the maximum value, in absolute term, is twice as large as for european options of comparable strike and maturity.
The dashboard in Figure 8.3 is generated with the following script:

```
> dtExpiry <- myDate("01jan2011")
> underlying <- "IBM"
> Strike <- 100
> barrier <- 160
> ba <- fInstrumentFactory("standardbarrier", quantity = 1, params = list(cp = "cuo",
     strike = Strike, barrier = barrier, rebate = 0, dtExpiry = dtExpiry,
     underlying = underlying, discountRef = "USD.LIBOR", trace = F))
> und.seq <- seq(40, 160, by = 5)
> vol.seq <- seq(0.1, 0.9, by = 0.1)
> base.env <- DataProvider()
> dtCalc <- myDate("01jan2010")
> setData(base.env, underlying, "DivYield", dtCalc, 0.02)
> setData(base.env, "USD.LIBOR", "Yield", dtCalc, 0.02)
> OptionDashBoard(ba, base.env, dtCalc, und.seq, vol.seq)
```

*Asian Option*

An asian option is an option on the average value of the underlying asset, computed over a period of time prior to expiry. This type is very popular in commodities markets, because it captures the price risk on a market participant that produces or consumes the asset at a steady rate over time. In spite of its apparent complexity, asian options are in fact fairly to manage. One can think of an Asian option as a regular European option, with an underlying asset that is the average value. The value of an asian call is lower than the value of a comparable european vanilla call, since the volatility of an average is lower than the volatility of the price at expiry. For the same reason, the vega of an Asian option is about half of the vega of its European conterpart.
The dashboard in Figure 8.4 is generated with the following script:

```
> dtExpiry <- myDate("01jan2011")
> underlying <- "IBM"
> Strike <- 100
> ba <- fInstrumentFactory("asian", quantity = 1, params = list(cp = "c",
     strike = Strike, dtExpiry = dtExpiry, dtStart = dtCalc, dtEnd = dtExpiry,
     underlying = underlying, discountRef = "USD.LIBOR", trace = F))
> und.seq <- seq(40, 160, by = 5)
> vol.seq <- seq(0.1, 0.9, by = 0.1)
> base.env <- DataProvider()
> dtCalc <- myDate("01jan2010")
> setData(base.env, underlying, "DivYield", dtCalc, 0.02)
> setData(base.env, "USD.LIBOR", "Yield", dtCalc, 0.02)
> OptionDashBoard(ba, base.env, dtCalc, und.seq, vol.seq)
```

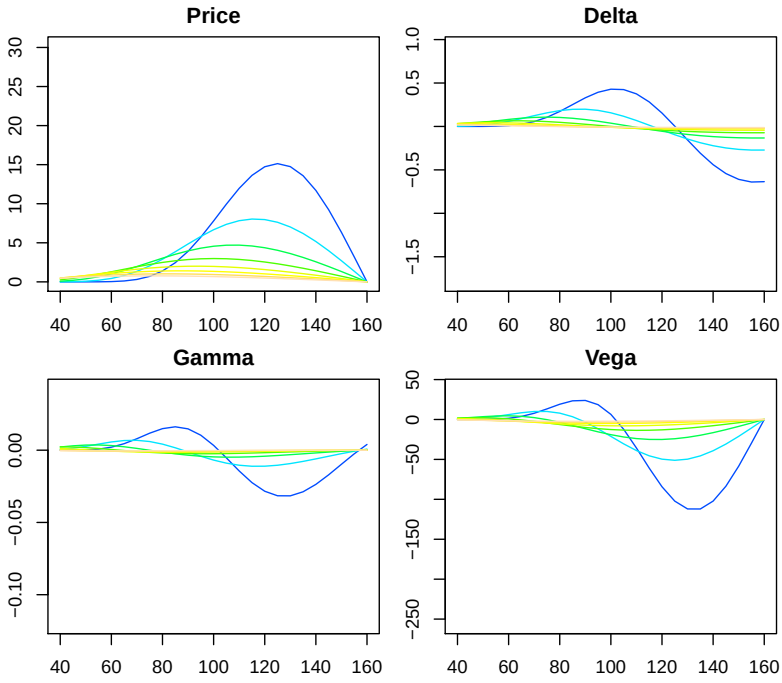



FIGURE 8.3: European Barrier Call (UAO) Strike=100

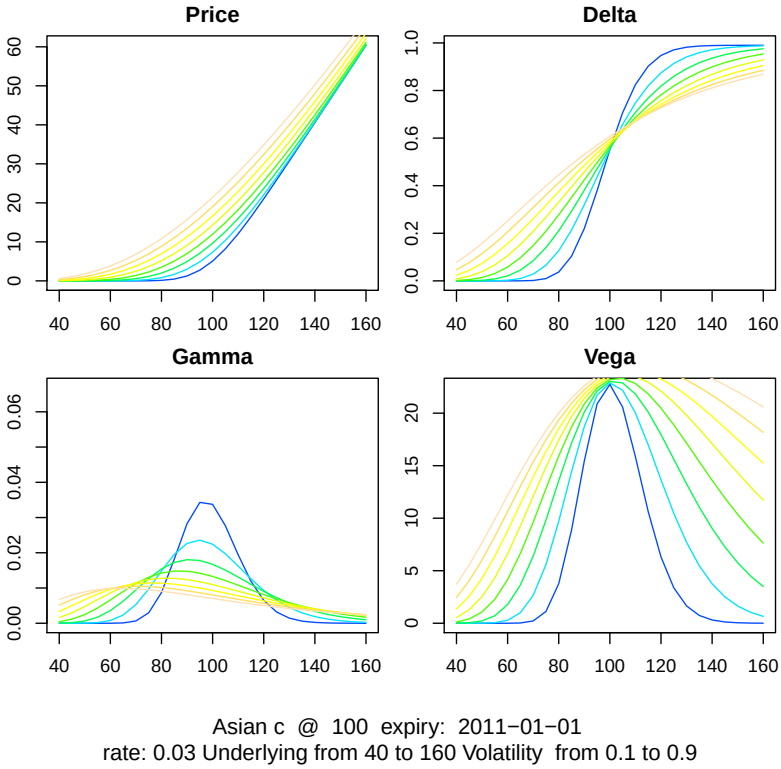



FIGURE 8.4: European Asian Call Strike=100

# CHAPTER 9

# DYNAMIC HEDGING IN THE BLACK-SCHOLES FRAMEWORK

```
> library(fInstrument)
> library(DynamicSimulation)
> library(empfin)
> library(gregmisc)
> library(BB)
> library(splines)
> library(SuppDists)
```

In this chapter, we investigate the effectiveness of hedging strategies in the Black-Scholes framework, and assess the impact of departures from the assumptions of the Black-Scholes model. Indeed, the ultimate measure of risk is the quality of the dynamic replicating strategy, and the distribution of the residual wealth at expiry of the option. Throughout this chapter, we place ourselves in the situation of a financial institution that sells a derivative and dynamically hedges the risk.

## 9.1 SELF-FINANCING REPLICATING PORTFOLIO

*Continuous rebalancing*

Let $C(S_t, t)$ be the price of a call option at time $t, 0 \leq t \leq T$, with the spot price being $S_t$. We first show that we can construct a trading strategy involving the underlying asset and a bank account, which will replicate exactly the option payoff $\max(S_T - K, 0)$ at time $T$ . This strategy is self-financing, meaning that it does not require any cash inflow, nor does it generate any cash outflow after $t = 0$. Because of this self-financing property, the value of the option must be equal to the initial value of this

replicating portfolio. Otherwise, a sure profit could be gained by trading the option against the replicating portfolio. The replicating portfolio $P$ is made of $\alpha_t$ shares and a notional amount $\beta_t$ of risk-free bond. The portfolio value is thus:

$$P_t = \alpha_t S_t + \beta_t B_t \tag{9.1}$$

where $B_t$ is the price of the risk-free bond at time $t$:

$$B_t = e^{-r(T-t)}$$

and $\alpha_t = \frac{\partial C}{\partial S}$. At $t = 0$ the portfolio has by definition the same value as the option:

$$P_0 = C(S_0, 0)$$

The residual wealth, $V_t$ is the difference between the replicating portfolio and the option:

$$V_t = P_t - C(S_t, t)$$

We show that in fact $V_t = 0,\ 0 \leq t \leq T$. Using the definition of the hedge portfolio, we have:

$$\begin{aligned} V_t &= \frac{\partial C}{\partial S} S_t + \beta_t B_t - C(S_t, t) \\ &= \Pi_t + \beta_t B_t \beta_t B_t \end{aligned}$$

Differentiate:

$$d V_t = d\Pi_t + \beta_t d B_t$$

From the derivation of the Black-Scholes equation (see Appendix or Hull (1997)), we know that the portfolio $\Pi_t$ is riskless:

$$d\Pi_t = r\Pi d t$$

therefore,

$$\begin{aligned} d V_t &= r\Pi_t d t + \beta_t r B d t \\ &= r V_t d t \end{aligned} \tag{9.2}$$

Since $V_0 = 0$, equation 9.2 has a unique solution which is:

$$V_t = 0,\ 0 \leq t \leq T$$

and we conclude, that portfolio $P$ replicates the option at all time, and in particular at expiry. The dynamic of the replication is governed by the assumptions of the Black-Scholes framework,which are recalled here:

- The underlying asset follows a geometric Brownian motion process, with constant and known volatility. The actual drift does not matter.
- The interest rate is constant.
- Trading is frictionless: there are no trading costs, shares can be purchase in arbitrary fractions, trading can take place continuously.

Clearly, none of these assumptions are verified in real life, and we expect that departure from these assumptions will affect the quality of the dynamic replication described by the Black-Scholes model. The purpose of this chapter is to investigate the effect of departures from these assumptions, and address the question: how useful is the Black-Scholes model in a environment where Black-Scholes assumptions are not verified?
We will adress this question by means of simulations that are now described.

*Discreet rebalancing*

Since trading does not occur continuously, a replicating strategy must consist in a decision rule that is implemented at regular intervals. In the case of the delta-hedging strategy the rule will be:

1. Define a rebalancing interval $\Delta t$. Trading dates with therefore be $\Delta t, 2\Delta t, \ldots, T - \Delta t$.
2. At each trading date $t$, buy or sell the underlying stock so that the hedge portfolio holds a quantity of risky asset equal to $\frac{\partial C_t}{\partial S_t}$.

To describe the process step by step, we use the following notation:

$C_t$ Value of derivative at time $t$

$V_t$ Value of hedge portfolio

$B_t$ Value of a pure discount bond maturing at T

$\alpha_t$ Delta of derivative, also the quantity of risky asset in the replicating portfolio

$\beta_t$ Quantity of riskless asset in hedge.

At $t = 0$, the derivative is sold at price $C_0$ and the proceeds are used to purchase a hedge portfolio. The initial hedge is $\alpha_0 S_0 + \beta_0 B_0$, where $\beta_0$ is computed from the acounting identity:

$$C_0 = \alpha_0 S_0 + \beta_0 B_0$$

At trading time $t$, the decision rule is as follows:

1. Compute the value of the hedge portfolio formed at the previous time step:

$$V_t = \alpha_{t-\Delta t} S_t + \beta_{t-\Delta t} B_t$$

2. Compute the amount of risky security to hold:

$$\alpha_t = \frac{\partial C_t}{\partial S_t}$$

3. Since the strategy must be self-financing, determine the quantity of bond to be lent or borrowed:

$$\beta_t = \frac{V_t - \alpha_t S_t}{B_t}$$

At expiry of the derivative, the bond is worth par, and the residual wealth is:

$$-C_T + \alpha_{T-\Delta t} S_T + \beta_{T-\Delta t}$$

The quality of a model is ultimately measured by the residual error: the liquidation value of the replicating portfolio, less the exercise value of the option: $V_T - C_T$.

*Illustration*

Let's illustrate this rebalancing process through a simple example that will also demonstrate the self-financing nature of the portfolio. A financial institution writes (sells) an at-the-money option on a stock that does not pay any dividend, and worth €100. The option expires in two months, and for illustrative purpose, the hedge will be rebalanced every week (in practice, of course, dynamic rebalancing is done much more often). Interest rate is 2% and volatility 30%.

```
> r <- 0.02
> T <- 2/12
> S0 <- 100
> K <- 100
> sigma <- 0.3
> p <- GBSOption(TypeFlag = "c", S = S0, X = K, Time = T, r = r,
    b = r, sigma)@price
> delta <- GBSGreeks("delta", TypeFlag = "c", S = S0, X = K, Time = T,
    r = r, b = r, sigma)
```

The initial hedge portfolio is constructed as follows:

- The value of the hedge portfolio is the value of the derivative: $P_0 = 5.04$.

- The delta of the option is:

$$\frac{\partial C_0}{\partial S_0} = 0.54$$

  Thus the hedge portfolio must also hold $\alpha_0 = 0.54$ units of stocks, for a cost of $\alpha_0 \times 100 = 53.52$.

- this position is funded by a riskless borrowing of $\beta_0 B = P_0 - 100\alpha_0 = -48.48$.

- Since the zero-coupon bond is worth $B_0 = e^{-rT} = 0.997$, we get $\beta_0 = -48.64$.

At every time step, the portfolio is adjusted acording to the algorithm of the previous section.
This calculation is easily performed with the supplied packages:

```
> dtStart <- myDate("01jun2010")
> dtObs <- dtStart
> dtExpiry <- myDate("01aug2010")
> underlying <- "IBM"
> K <- 100
> sigma <- 0.3
> a <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
     strike = K, dtExpiry = dtExpiry, underlying = underlying,
     discountRef = "USD.LIBOR", trace = FALSE))
> base.env <- DataProvider()
> setData(base.env, underlying, "Price", dtObs, 100)
> setData(base.env, underlying, "DivYield", dtObs, 0)
> setData(base.env, underlying, "ATMVol", dtObs, sigma)
> setData(base.env, underlying, "discountRef", dtObs, "USD.LIBOR")
> setData(base.env, "USD.LIBOR", "Yield", dtObs, 0.02)
```

At this stage, we can price the asset with

```
> p <- getValue(a, "Price", dtStart, base.env)
```

which gives a value of $p = 4.87$. Next, define the parameters to simulate a dynamic hedging policy with weekly rebalancing and two price paths:

```
> nbSteps <- 8
> nbPaths <- 2
> dtSim <- seq(dtObs, dtExpiry, length.out = nbSteps + 1)
```

The simulated price path is generated by

```
> tSpot <- pathSimulator(dtSim = dtSim, nbPaths = nbPaths, innovations.gen = sobolInnovations,
     path.gen = logNormal, path.param = list(mu = 0, sigma = sigma),
     S0 = 100, antithetic = F, standardization = TRUE, trace = F)
```

and the simulated paths are placed in a new data provider:

```
> sce.env <- DataProvider(parent = base.env)
> setData(sce.env, underlying, "Price", time(tSpot), as.matrix(tSpot))
```

We can now run the delta-hedge strategy along each path:

```
> assets = list(a)
> res <- deltaHedge(assets, sce.env, params = list(dtSim = time(tSpot),
      transaction.cost = 0), trace = F)
```

The result is a data structure that containts the residual wealth (hedging error) per scenario.
Tables 9.1 and 9.2 illustrate how the hedge portfolios evolve over time, based on the path of the underlying asset. In Table 9.2, the option expires out of the money, and the hedge portfolio ends up being exclusively cash. In Table 9.1, the option expires in the money, and the hedge portfolio holds at expiry one unit of stock. Note the amount of leverage required by delta hedging when the option expires in the money: the option payoff is 45 cents, and this is replicated by a portfolio holding €100.45 worth of stock, funded by borrowing €99.00.
The data shown in both tables are extracted from the simulation results and formated with the following function.

```
> myTable <- function(iScenario, res, ...) {
      tSpot <- res$spot
      nbSteps <- dim(tSpot)[1]
      z <- matrix(nrow = nbSteps, ncol = 6)
      z[, 1] <- seq(1, nbSteps)
      z[, 2] <- tSpot[, iScenario]
      z[, 3] <- res$stock[, iScenario]
      z[, 4] <- res$price[, iScenario]
      z[, 5] <- res$bond[, iScenario]
      z[, 6] <- res$portfolio[, iScenario]
      colnames(z) <- c("time", "stock price", "delta", "option",
          "bond pos", "hedge port.")
      xtable(z, ...)
  }
> tab.1 <- myTable(1, res)
> tab.2 <- myTable(2, res)
```

*A More Realistic Example*

More time steps and scenarios are needed in order to accurately assess the distribution of residual wealth. The next experiment uses the parameters of the previous section, with 200 time steps and 1000 sceanrios.
Figure 9.1 displays a few simulated paths for the underlying asset, and an histogram of the residual wealth.

```
> nbSteps <- 200
> nbPaths <- 1000
> dtSim <- seq(dtObs, dtExpiry, length.out = nbSteps + 1)
> tSpot <- pathSimulator(dtSim = dtSim, nbPaths = nbPaths, innovations.gen = sobolInnovations,
```

| | time | stock price | delta | option | bond pos | hedge port. |
|---|---|---|---|---|---|---|
| 1 | 1.00 | 100.00 | 0.52 | 4.87 | -47.56 | 4.87 |
| 2 | 2.00 | 99.98 | 0.52 | 4.55 | -47.36 | 4.84 |
| 3 | 3.00 | 102.84 | 0.62 | 5.85 | -57.88 | 6.31 |
| 4 | 4.00 | 96.95 | 0.39 | 2.47 | -35.48 | 2.62 |
| 5 | 5.00 | 90.42 | 0.13 | 0.50 | -11.87 | 0.04 |
| 6 | 6.00 | 98.69 | 0.44 | 2.36 | -42.81 | 1.13 |
| 7 | 7.00 | 96.99 | 0.32 | 1.20 | -30.68 | 0.35 |
| 8 | 8.00 | 103.41 | 0.79 | 3.96 | -78.97 | 2.39 |
| 9 | 9.00 | 104.79 | 1.00 | 4.79 | -101.35 | 3.44 |

Table 9.1: Delta-hedging of a Europen call, strike: 100

| | time | stock price | delta | option | bond pos | hedge port. |
|---|---|---|---|---|---|---|
| 1 | 1.00 | 100.00 | 0.52 | 4.87 | -47.56 | 4.87 |
| 2 | 2.00 | 95.63 | 0.37 | 2.62 | -32.80 | 2.57 |
| 3 | 3.00 | 97.67 | 0.43 | 3.12 | -38.97 | 3.31 |
| 4 | 4.00 | 101.70 | 0.59 | 4.80 | -54.74 | 5.03 |
| 5 | 5.00 | 99.53 | 0.49 | 3.22 | -45.59 | 3.74 |
| 6 | 6.00 | 97.56 | 0.38 | 1.89 | -34.85 | 2.74 |
| 7 | 7.00 | 95.43 | 0.23 | 0.77 | -20.24 | 1.91 |
| 8 | 8.00 | 92.34 | 0.03 | 0.05 | -2.02 | 1.19 |
| 9 | 9.00 | 94.01 | 0.00 | 0.00 | 1.24 | 1.24 |

Table 9.2: Delta-hedging of a Europen call, strike: 100

```
      path.gen = logNormal, path.param = list(mu = 0, sigma = sigma),
      S0 = 100, antithetic = F, standardization = TRUE, trace = F)
> sim.env <- DataProvider(parent = base.env)
> setData(sim.env, underlying, "Price", time(tSpot), as.matrix(tSpot))
> sim.deltaHedge <- deltaHedge(assets, sim.env, params = list(dtSim = time(tSpot),
      transaction.cost = 0), trace = F)
```

As expected, the distribution of residual wealth is centered, and has a standard deviation of

```
> sd.wealth <- sd(t(tail(sim.deltaHedge$wealth, 1)))
```

A confidence interval (say, 80%) for the P&L of the delta-hedging strategy is readily computed:

```
> pl.confidence <- qnorm(0.2, mean = 0, sd = sd.wealth)
```

which gives a value of $-0.33$. In other words, there is a 80% probability that this delta-hedging strategy will yield a P&L greater than $-0.33$ per € of nominal. In the next couple of sections, we will provide a more rigorous analysis of these observations.

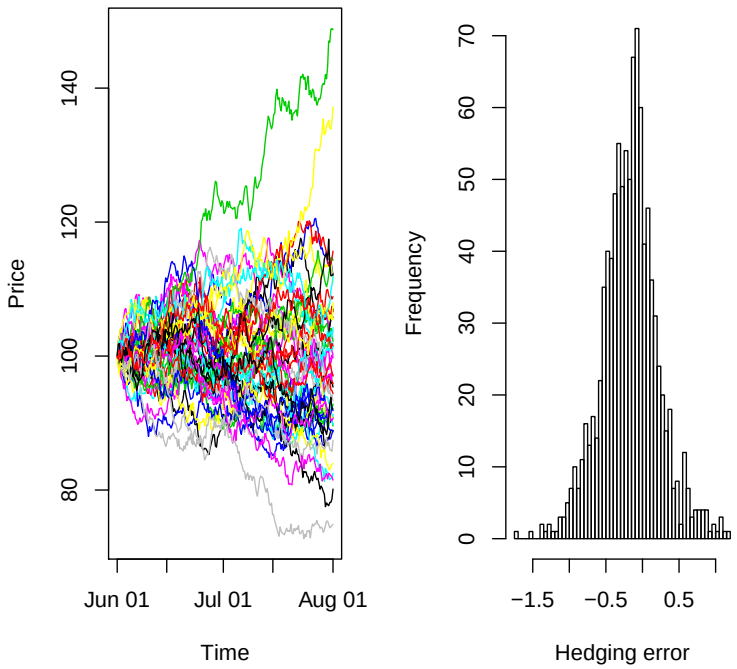



FIGURE 9.1: Simulation of a delta-hedging strategy. Left panel: the log-normal underlying price process. Right panel: distribution of the hedging error

With this simulation tool, we will now consider the effect on hedging effectiveness of:

- discrete rather than continuous hedging,
- transaction costs, and
- unknown and stochastic volatility.

## 9.2 HEDGING ERROR DUE TO DISCRETE HEDGING

We first consider the risk related to discreet hedging. The dynamic of the replicating portfolio, $V_t$ rebalanced at times $t_i, i = 1, \ldots, N$. For $t \in [t_i, t_{i+1})$ is:

$$dV_t = \frac{\partial C(t_i, S_{t_i})}{\partial S} dS_t \tag{9.3}$$

The dynamic of the option price is:

$$dC_t = \frac{\partial C(t,S_t)}{\partial S} dS_t$$

The difference between the two processes being that in the hedge portfolio, the quantity of stock ${}^{\partial C(t_i,S_{t_i})}/_{\partial S_{t_i}}$ (Eq. 9.3), is held constant between two rebalancing dates.
The hedging error process $\epsilon_t = C_t - V_t$ is therefore:

$$d\epsilon_t = \left( \frac{\partial C(t,S_t)}{\partial S} - \frac{\partial C(t_i,S_{t_i})}{\partial S} \right) dS_t$$

It can be shown that:

$$\lim_{h \to 0} E^Q \left[ \frac{\epsilon_T^2}{\Delta t} \right] = E^Q \left[ \frac{1}{2} \int_0^T \left( \frac{\partial^2 C(S_t,t)}{\partial S_t^2} \right)^2 S_t^4 \sigma^4 dt \right]$$

It can also be shown that the process $\epsilon_t$ converges in distribution to a process that has the following expression:

$$\frac{\epsilon_t}{\sqrt{\Delta t}} \to \frac{1}{\sqrt{2}} \int_0^t \frac{\partial^2 C(S_u,u)}{\partial S^2} \sigma^2 S_u^2 dZ_u$$

This expression highlights the following points:

1. The variance of the hedging is inversely related to the hedging frequency, and
2. is directly related to the magnitude of $\frac{\partial^2 C(S_u,u)}{\partial S^2}$, which is the Gamma of the option.

*Empirical Test*

**Effect of Hedging Frequency** To test the impact of hedging frequency on the distribution of delta-hedging error, we simulate the dynamic delta hedging of a ATM option. The price paths are generated with the same volatility as the one used for pricing, and there are no transaction costs. Therefore, the only source of hedging error is the discrete hedging policy. The simulation involves several steps that are outlined next.
First, define the derivative to be dynamically hedged:

```
> dtExpiry <- mytDate("01jan2011")
> underlying <- "IBM"
> K <- 100
> a <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
      strike = K, dtExpiry = dtExpiry, underlying = underlying,
      discountRef = "USD.LIBOR", trace = F))
```

Next, we define the market data that we be used in the simulation. The base environment holds data that will not change from one simulation to the next.

```
> dtStart <- mytDate("01jan2010")
> nbPaths <- 1000
> sigma <- 0.3
> base.env <- DataProvider()
> setData(base.env, underlying, "Price", dtStart, 100)
> setData(base.env, underlying, "DivYield", dtStart, 0.02)
> setData(base.env, underlying, "ATMVol", dtStart, sigma)
> setData(base.env, underlying, "discountRef", dtStart, "USD.LIBOR")
> setData(base.env, "USD.LIBOR", "Yield", dtStart, 0.02)
```

We next perform a series of dynamic hedging simulations, while varying the number of time steps.

```
> nb.range <- seq(10, 500, by = 50)
> mean.error <- 0 * nb.range
> sd.error <- 0 * nb.range
> for (i in seq_along(nb.range)) {
    nbSteps <- nb.range[i]
    dtSim <- seq(as.timeDate(dtStart), as.timeDate(dtExpiry),
        length.out = nbSteps + 1)
    tSpot <- pathSimulator(dtSim = dtSim, nbPaths = nbPaths,
        innovations.gen = sobolInnovations, path.gen = logNormal,
        path.param = list(mu = 0, sigma = sigma), S0 = 100, antithetic = F,
        standardization = TRUE, trace = F)
    sce.env <- DataProvider(parent = base.env)
    setData(sce.env, underlying, "Price", time(tSpot), as.matrix(tSpot))
    assets = list(a)
    res <- deltaHedge(assets, sce.env, params = list(dtSim = time(tSpot),
        transaction.cost = 0), trace = F)
    expiry.error <- tail(res$wealth, 1)
    mean.error[i] <- mean(expiry.error)
    sd.error[i] <- sd(as.vector(expiry.error))
 }
```

Figure 9.2 illustrates the mean and standard deviation of the hedging error as a function of the number of rehedging dates. The figure also shows the curve

$$y = a + b\frac{1}{\sqrt{N}}$$

fitted to the standard deviation of the hedging error. The theory predicts that the standard deviation of the hedging error is proportional to $\frac{1}{\sqrt{N}}$, where $N$ is the number of re-hedging dates.

```
> plotCI(x = nb.range, y = mean.error, uiw = sd.error/2, liw = sd.error/2,
    ylim = c(min(mean.error - sd.error/2), max(mean.error + sd.error/2)),
    xlab = "Nb of rebalancings", ylab = "Hedging error")
> q <- lm((mean.error + sd.error/2) ~ sqrt(1/nb.range))
> yy <- predict(interpSpline(nb.range, q$fitted.values))
> lines(yy, lwd = 2, col = "red")
```

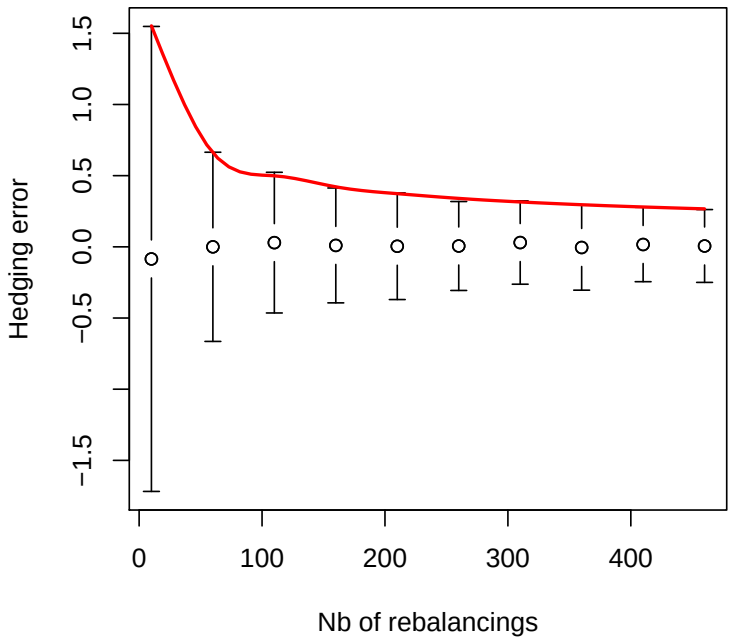



FIGURE 9.2: Delta hedging error vs. hedging frequency

**Effect of Gamma: Vanilla vs. Binary** To investigate the effect of gamma on hedging error, we will compare the effectiveness of delta hedging, applied to two options that initially have the same price and delta. The initial Gammas are within 10% of each other. However, the options differ substantially in one aspect: the first option is a European call, and its Gamma varishes when the option gets deep in the money, while the second option is a binary call, whose Gamma remains large and actually changes sign under the same conditions.

To perform the experiment, we first define the data environment: the spot price is 100, option maturity will be 1 year.

```
> dtExpiry <- myDate("01jan2011")
> dtStart <- myDate("01jan2010")
> underlying <- "IBM"
> sigma <- 0.3
> base.env <- DataProvider()
> setData(base.env, underlying, "Price", dtStart, 100)
```

```
> setData(base.env, underlying, "DivYield", dtStart, 0.02)
> setData(base.env, underlying, "ATMVol", dtStart, sigma)
> setData(base.env, underlying, "discountRef", dtStart, "USD.LIBOR")
> setData(base.env, "USD.LIBOR", "Yield", dtStart, 0.02)
```

The strike of the European option is set to $K = 130$.

```
> K <- 130
> op.eu <- fInstrumentFactory("vanilla", quantity = 1, params = list(cp = "c",
     strike = K, dtExpiry = dtExpiry, underlying = underlying,
     discountRef = "USD.LIBOR", trace = F))
> P <- getValue(op.eu, "Price", dtStart, base.env)
> D <- getValue(op.eu, "Delta", dtStart, base.env)
```

The price is $P = 3.99$, delta is $D = 0.26$.
The strike and payoff of the binary option are set in order to match the price and delta of the European call, so as to make a meaningful comparison of hedging error. This involves solving a set of two non-linear equations (price and delta) in two unknown (payoff and strike). These equations are solved by means of the following function:

```
> price.delta.error <- function(x, base.env) {
     Q <- x[1]
     K.b <- x[2]
     op.bi <- fInstrumentFactory("binary", quantity = Q, params = list(cp = "c",
         strike = K.b, dtExpiry = dtExpiry, underlying = underlying,
         discountRef = "USD.LIBOR", trace = F))
     Pa <- getValue(op.bi, "Price", dtStart, base.env)
     Da <- getValue(op.bi, "Delta", dtStart, base.env)
     f <- rep(NA, 2)
     f[1] <- (P - Pa)
     f[2] <- (D - Da)
     f
 }
> ans <- dfsane(par = c(100, 100), fn = price.delta.error, base.env = base.env)

Iteration:  0  ||F(x0)||:  31.383
iteration:  10  ||F(xn)|| =   0.016648
iteration:  20  ||F(xn)|| =   0.11474
iteration:  30  ||F(xn)|| =   0.042438
iteration:  40  ||F(xn)|| =   0.0090054
iteration:  50  ||F(xn)|| =   0.0013793
iteration:  60  ||F(xn)|| =   6.25e-06

> Q <- ans$par[1]
> K.b <- ans$par[2]
```

The solution is a binary call with payoff $Q = 58.45$ and strike $K = 155.06$.

```
> op.bi <- fInstrumentFactory("binary", quantity = Q, params = list(cp = "c",
     strike = K.b, dtExpiry = dtExpiry, underlying = underlying,
     discountRef = "USD.LIBOR", trace = F))
> G.bi <- getValue(op.bi, "Gamma", dtStart, base.env)
> G.eu <- getValue(op.eu, "Gamma", dtStart, base.env)
```

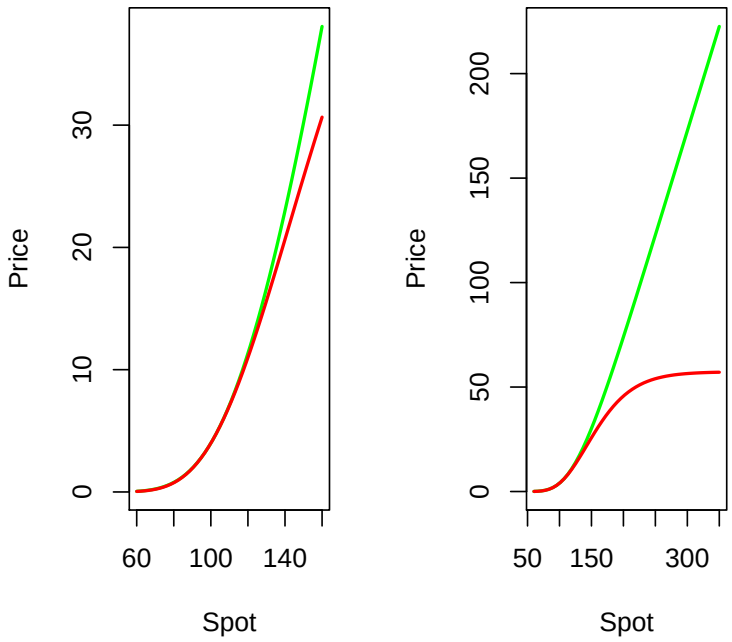



FIGURE 9.3: Price of Vanilla and Binary Calls. The left panel focuses around the initial spot price: locally, the the two options are almost identical. The difference between the two options appear in the right panel, for deep in-the-money scenarios

The Gammas of the two options are not identical, but within 10% of each others: 0.0107 for European option, 0.01 for the binary option.
To illustrate the similarity between the two derivatives, we plot in Figure9.3 the prices of both options for a range of underlying values, with a zoom around the current spot price.

```
> sce.1 <- t(as.matrix(seq(60, 160, length.out = 50)))
> sce.2 <- t(as.matrix(seq(60, 350, length.out = 50)))
> sim.env.1 <- DataProvider(parent = base.env)
> setData(sim.env.1, underlying, "Price", dtStart, sce.1)
> sim.env.2 <- DataProvider(parent = base.env)
> setData(sim.env.2, underlying, "Price", dtStart, sce.2)
> p.eu.1 <- getValue(op.eu, "Price", dtStart, sim.env.1)
> p.bi.1 <- getValue(op.bi, "Price", dtStart, sim.env.1)
> p.eu.2 <- getValue(op.eu, "Price", dtStart, sim.env.2)
> p.bi.2 <- getValue(op.bi, "Price", dtStart, sim.env.2)
```

A delta hedging strategy is next simulated for both derivatives, using 100 steps, 5000 paths.

```
> nbSteps <- 100
> nbPaths <- 5000
> dtSim <- seq(as.timeDate(dtStart), as.timeDate(dtExpiry), length.out = nbSteps +
      1)


> tSpot <- pathSimulator(dtSim = dtSim, nbPaths = nbPaths, innovations.gen = sobolInnovations,
      path.gen = logNormal, path.param = list(mu = 0, sigma = sigma),
      S0 = 100, antithetic = F, standardization = TRUE, trace = F)
> sce.env <- DataProvider(parent = base.env)
> setData(sce.env, underlying, "Price", time(tSpot), as.matrix(tSpot))
> assets = list(op.eu)
> res1 <- deltaHedge(assets, sce.env, params = list(dtSim = time(tSpot),
      transaction.cost = 0), trace = F)
> assets = list(op.bi)
> res2 <- deltaHedge(assets, sce.env, params = list(dtSim = time(tSpot),
      transaction.cost = 0), trace = F)
```

Figure 9.4 displays the histograms of residual errors.

```
> par(mfrow = c(1, 2))
> hist(tail(res1$wealth, 1), 50, xlab = "wealth", main = "Vanilla Call")
> hist(tail(res2$wealth, 1), 50, xlab = "wealth", main = "Binary Call")
> par(mfrow = c(1, 1))
```

Statistics on the hedging error at expiry are summarized in Table 9.3. The standard deviation of the binary hedging error is about three times larger than for the vanilla option. Most importantly, the excess kurtosis is almost 10 times larger, indicating the possibilily that applying a standard delta-hedging strategy to a binary option can result in extreme losses (or gains!). All of this in a context where the only departure from the Black-Scholes assumptions is a discrete rather than continuous hedging. We next investigate the impact of transaction costs.

```
> w1 <- as.vector(tail(res1$wealth, 1))
> w2 <- as.vector(tail(res2$wealth, 2))
> m1 <- moments(w1)
> m2 <- moments(w2)
> xm <- xtable(data.frame(cbind(Vanilla = m1, Binary = m2)), display = c("s",
      "e", "e"))
```

## 9.3 HEDGING ERROR DUE TO TRANSACTION COST

Let $S_t$ be the mid price of the underlying asset and $k$ the proportional transaction cost: the hedger buys at the ask price:

$$S_t^a = S_t(1+\frac{k}{2})$$

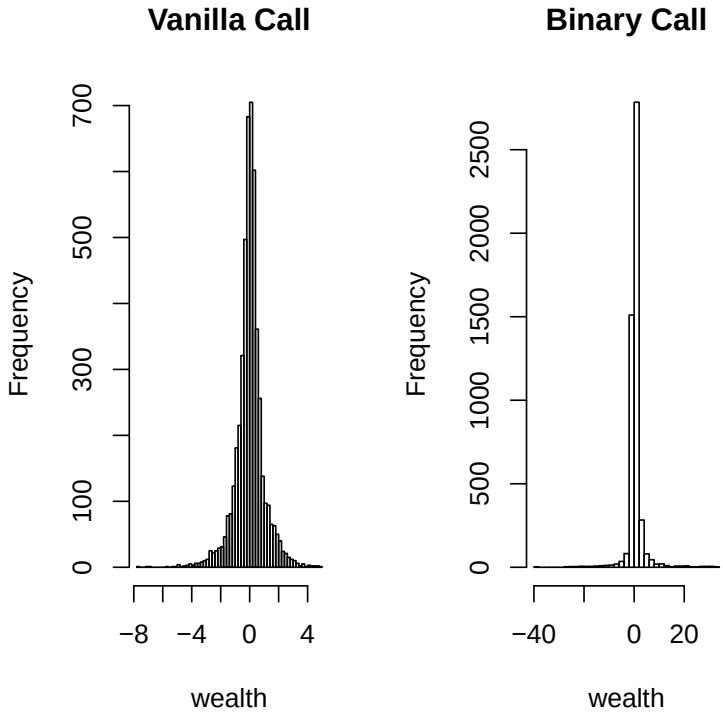



FIGURE 9.4: Delta Hedging Errors: Vanilla and Binary Calls

| | Vanilla | Binary |
|---|---|---|
| mean | -5.06e-03 | 4.55e-01 |
| sigma | 9.96e-01 | 2.85e+00 |
| skew | -4.81e-01 | 8.35e-01 |
| kurt | 5.21e+00 | 4.44e+01 |

Table 9.3: Moments of hedging error: Vanilla vs. Binary options

and sells at the bid:

$$S_t^a = S_t(1 - \frac{k}{2})$$

Clearly, transaction costs increase the cost of replication. To compensate for that, one needs to start with a higher budget, that is, the option seller should change a higher option premium than the one given by the Black-Scholes model. This higher option premium can be obtained in the Black-Scholes framework by increasing volatility. Leland (1985) shows that one can adjust the Black-Scholes volatility as follows:

$$\sigma^{+2} = \sigma^2 \left(1 + \sqrt{\frac{2}{\pi}} \frac{k}{\sigma\sqrt{\Delta t}}\right) \tag{9.4}$$

where $\sigma$ is the Black-Scholes volatility, and $\sigma+$ the volatility adjusted for transaction costs.

*Empirical Test*

To test Leland's formula, we simulate the delta-hedging of a at-the-money European option, struck at 100, for transaction costs $k$ ranging from 0 to 1%. The simulation is performed as follows:

```
> tr.range <- seq(0, 1, 0.1)/100
> nbSteps <- 250
> dtSim <- seq(as.timeDate(dtStart), as.timeDate(dtExpiry), length.out = nbSteps +
      1)
> dt <- as.real(dtExpiry - dtStart)/(365 * nbSteps)
> tSpot <- pathSimulator(dtSim = dtSim, nbPaths = nbPaths, innovations.gen = sobolInnovations,
      path.gen = logNormal, path.param = list(mu = 0, sigma = sigma),
      S0 = 100, antithetic = F, standardization = TRUE, trace = F)
> sce.env <- DataProvider(parent = base.env)
> setData(sce.env, underlying, "Price", time(tSpot), as.matrix(tSpot))
> mean.error <- 0 * tr.range
> sd.error <- 0 * tr.range
> for (i in seq_along(tr.range)) {
      assets = list(a)
      res <- deltaHedge(assets, sce.env, params = list(dtSim = time(tSpot),
          transaction.cost = tr.range[i]/2), trace = F)
      expiry.error <- tail(res$wealth, 1)
      mean.error[i] <- mean(expiry.error)
      sd.error[i] <- sd(as.vector(expiry.error))
  }
```

We next compare the expected hedging error to the value predicted by Leland's formula. To do so, we convert the expected hedging error into a volatility adjustment, using the first order approximation:

$$P(\sigma) = P(\sigma^*) + (\sigma - \sigma^*)\frac{\partial P}{\partial \sigma}$$

where $P(\sigma)$ is the price of the option with volatility $\sigma$.

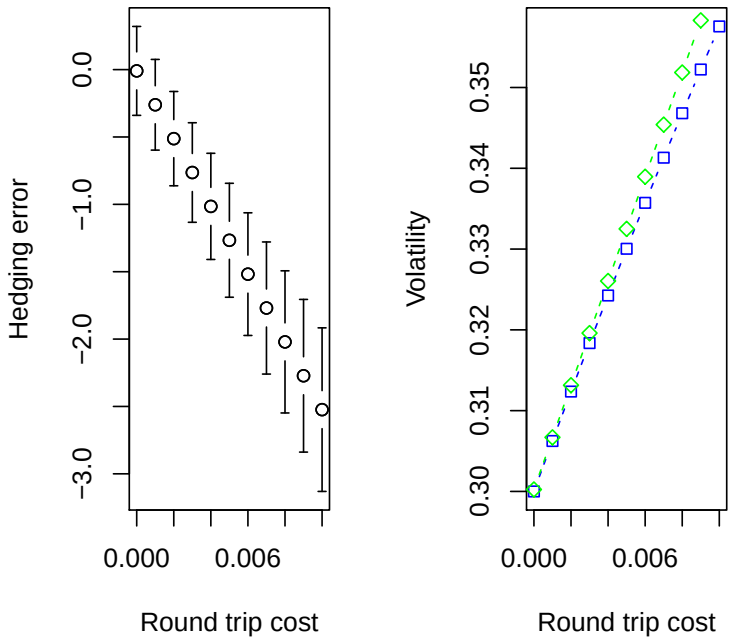



FIGURE 9.5: Delta hedging error vs. transaction cost. The left panel shows the mean and standard deviation of the hedging error, as a function of transaction cost. The right panel compares the empirical volatility adjustment (green)) to theoretical value from Leland's formula (blue)), given a Black-Scholes volatility of 30%.

```
> sigma.leland <- sigma * sqrt(1 + sqrt(2/pi) * (tr.range/(sigma *
      sqrt(dt))))
> vega <- getValue(a, "Vega", dtStart, base.env)
> sigma.sim <- sigma - mean.error/vega
```

The results are presented in Figure 9.5. We find very good agreement between the empirical result and Leland's formula.

## 9.4 HEDGING ERROR DUE TO UNKNOWN VOLATILITY

The strongest assumption of the Black-Scholes model is arguably the assumption that volatility is known and constant. we next investigate the impact of unknown volatility of hedging error, following a landmark paper by Karoui, Jeanblanc-Picquè & Shreve (1998).

To simplify the notation, assume now that hedging is done continuously, but that the volatility of the underlying asset is unknown. The dynamic of the underlying asset is:

$$dS_t = \mu dt + \sigma_t dW_t$$

The derivative is priced with an estimated volatility $\Sigma$.
The dynamic of the hedge portfolio is:

$$dV_t = \frac{\partial C_t^{\Sigma}}{\partial S_t} dS_t$$

By Itô's lemma:

$$dC(S_t, t) = \left( \frac{\partial C}{\partial t} + \frac{1}{2} \frac{\partial^2 C}{\partial S^2} \sigma_t^2 S_t^2 \right) dt + \frac{\partial C}{\partial S} dS_t$$

The price of the derivative verifies the Black-Scholes EDP:

$$\frac{\partial C}{\partial t} + \frac{1}{2} \Sigma^2 S_t^2 \frac{\partial^2 C}{\partial S^2} = 0$$

this yields,

$$d\epsilon_t = dV_t - dC_t = \frac{1}{2} \left[ \Sigma^2 - \sigma_t^2 \right] \frac{\partial^2 C}{\partial S^2} S^2 dt$$

and the hedging error at expiry is thus,

$$\epsilon_T = \frac{1}{2} \int_0^T \left[ \Sigma^2 - \sigma_t^2 \right] \frac{\partial^2 C}{\partial S^2} S^2 dt$$

We identify three components in the above equation:

1. the nature of the option, characterized by its Gamma.
2. the behavior of the market, characterized by $\sigma_t$,
3. the model calibration, summarized here by the Black-Scholes volatility $\Sigma$

If the Gamma keeps a constant sign, we can compute the fair value of the Black-Scholes volatility that sets the expected hedging error to 0 is

$$\sigma_{BS}^2 = \frac{E \int_0^T \Gamma S^2 \frac{\Delta S}{S}^2}{E \int_0^T \Gamma_{BS} S^2 dt}$$

Equation (**??**) also shows that, when the gamma keeps a constant sign, we can choose a pricing volatility $\Sigma$ such that $E(\epsilon_T)$ is positive.

| Payoff | Is BS Adapted? |
|---|---|
| Vanilla | Yes |
| European Binary | No |
| Up and Out Barrier | No |
| Asian | Yes |

Table 9.4: Fitness of the Black-Scholes Model

## 9.5 CONCLUSION

The experiments of this chapter highlight the crucial role of Gamma in dynamic hedging. In short, the original Black-Scholes framework remains adapted for options for which the Gamma does not change sign. This is summarized in Table 9.4. In such case, a positive expected hedging error can be ensured by setting the pricing volatility sufficiently high or low, as the case may be. Moreover, the Gamma of these options often vanishes in deep in the money or out of the money scenarios, and this renders of course the delta hedging strategy very effective. This may explain why the Black-Scholes model remains so widely used, despite its well documented shortcomings. Next chapter describes a framework for dealing with Gamma risk, while remainning in the Black-Scholes framework.

# CHAPTER 10

# VANNA-VOLGA PRICING AND HEDGING

*If you want to know the value of a security, use the price of another security that's similar to it. All the rest is strategy.*
E. Derman

This chapter presents a practical method for pricing and hedging derivatives, taking into account an uncertain volatility. This method is very popular for Foreign Exchange derivatives, but beyond that it illustrates an important principle of asset pricing, which is to relate the price of a complex derivative to the known price of simpler, liquid instruments.

## 10.1 PRINCIPLE

Assume a Ito dynamic for volatility, and consider a derivative security $O(t)$, which is now subject to two sources of randomness: the underlying asset $S_t$ and the volatility $\sigma_t$. An extended version of Ito's lemma give:

$$\begin{aligned} dO(t,K) &= \frac{\partial O}{\partial t}dt + \frac{\partial O}{\partial S}dS_t + \frac{\partial O}{\partial \sigma}d\sigma_t \\ &\quad + \frac{1}{2}\frac{\partial^2 O}{\partial S^2}(dS_t)^2 + \frac{1}{2}\frac{\partial^2 O}{\partial \sigma^2}(d\sigma_t)^2 + \frac{\partial^2 O}{\partial S \partial \sigma}dS_t d\sigma_t \end{aligned}$$

We now construct a portfolio made of option $O(t,K)$, and a hedge made of $\Delta_t$ units of the underlying asset and 3 delta-hedged vanilla options $C_i(t,K_i)$:

$$
\begin{aligned}
dO(t,K) - \Delta_t dS_t - \sum_{i=1}^{3} x_i dC_i(t,K_i) = \\
&\left[\frac{\partial O}{\partial t} - \sum_i x_i \frac{\partial C_i}{\partial t}\right] dt \\
&+ \left[\frac{\partial O}{\partial S} - \Delta_t - \sum_i x_i \frac{\partial C_i}{\partial S}\right] dS_t \\
&+ \left[\frac{\partial O}{\partial \sigma} - \sum_i x_i \frac{\partial C_i}{\partial \sigma}\right] d\sigma_t \\
&+ \frac{1}{2}\left[\frac{\partial^2 O}{\partial S^2} - \sum_i x_i \frac{\partial^2 C_i}{\partial S^2}\right] (dS_t)^2 \\
&+ \frac{1}{2}\left[\frac{\partial^2 O}{\partial \sigma^2} - \sum_i x_i \frac{\partial^2 C_i}{\partial \sigma^2}\right] (d\sigma_t)^2 \\
&+ \left[\frac{\partial^2 O}{\partial S \partial \sigma} - \sum_i x_i \frac{\partial^2 C_i}{\partial S \partial \sigma}\right] dS_t d\sigma_t
\end{aligned}
$$

We can choose $\Delta_t$ and $x_i$ to zero out the terms $dS_t, d\sigma_t, (d\sigma_t)^2$ and $dS_t d\sigma_t$. We are left with:

$$
\begin{aligned}
dO(t,K) - \Delta_t dS_t - \sum_{i=1}^{3} x_i dC_i(t,K_i) = \\
&\left[\frac{\partial O}{\partial t} - \sum_i x_i \frac{\partial C_i}{\partial t}\right] dt \\
&+ \frac{1}{2}\left[\frac{\partial^2 O}{\partial S^2} - \sum_i x_i \frac{\partial^2 C_i}{\partial S^2}\right] (dS_t)^2
\end{aligned}
$$

Using the definition of $dS_t$ and retaining the first-order terms, we have, for all securities function of $S_t$:

$$
\frac{\partial f}{\partial t} dt + \frac{1}{2}\frac{\partial^2 f}{\partial S^2} = \left(\frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2} S_t^2 \sigma^2\right) dt
$$

Moreover, each asset satisfies the Black-Scholes PDE:

$$
\frac{\partial f}{\partial t} + r S_t \frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf \tag{10.1}
$$

For each asset, $\frac{\partial f}{\partial S} = 0$, and we get:

$$\frac{\partial f}{\partial t}dt+\frac{1}{2}\frac{\partial^2 f}{\partial S^2}=rfdt$$

The hedged portfolio is thus riskless:

$$dO(t,K)-\Delta_t dS_t-\sum_{i=1}^{3}x_i dC_i(t,K_i)=$$
$$r\left[O(t,K)-\sum_i x_i C_i(t,K_i)\right]dt$$

In summary, we can still have a locally perfect hedge when volatility is stochastic, as long as the prices $O(t,K)$ and $C_I(t,K_i)$ follow the Black-Scholes equation.

Discretize equation 10.2 at time $t=T-\delta t$, where $T$ is the option expiry. We get:

$$O(T,K)-O(t,K)-\Delta_t(S_T-S_t)-\sum_i x_i(t)\left[C_i(T,K_i)-C_i(t,K_i)\right]=$$
$$r\left[O(t,K)-\Delta_t S_t-\sum_i x_i(t)C(t,K_i)\right]\delta t$$

Up to now, we have assumed that all assets are priced with the same volatility $\sigma_t$. In practice, we observe market prices $C_i^M(t,K_i)$ that differ from the theoretical Black-Scholes prices $C_i(t,K_i)$. The hedge described above must be implemented with options traded at market price. What are the implications for the price $O(t,K)$? An approximate argument is presented below; see Shkolnikov (2009) for a rigorous treatment.

Consider a hedged position at time $T-\delta t$: long the exotic derivative $O$ with market value $O^M(t)$ and short the hedge portfolio. The value of the hedged position at expiry is:

$$W_T=O(T)-\Delta_t S_T-\sum_i x_i C_i(T)-r\left(O^M(t)-\sum_i x_i C_i^M(t)-\Delta S_t\right)\delta t$$

Set

$$O^M(t,K)=O(t,K)+\sum_i x_i\left[C_i^M(t,K_i)-C_i(t,K_i)\right] \tag{10.2}$$

to obtain $W_T=0$.

In summary, if we have a wealth $O^M(t,K)$ defined by 10.2 at time $T-\delta t$, then we can replicate the payoff $O(T,K)$ at expiry, with a hedge at market price. The argument made on the interval $[T-\delta t,T]$ can be applied by backward recursion for each time interval until $t=0$.

We have both a hedging strategy and a process for adjusting the price of any derivative to account for the smile. let's now consider some implementation details.

## 10.2 IMPLEMENTATION

The weights $x_i$ are obtained by solving the system of linear equations:

$$\begin{aligned}\frac{\partial O}{\partial \sigma} &= \sum_i x_i \frac{\partial C_i}{\partial \sigma} \\ \frac{\partial^2 O}{\partial \sigma^2} &= \sum_i x_i \frac{\partial^2 C_i}{\partial \sigma^2} \\ \frac{\partial^2 O}{\partial S \partial \sigma} &= \sum_i x_i \frac{\partial^2 C_i}{\partial S \partial \sigma}\end{aligned}$$

or,

$$b = Ax \tag{10.3}$$

Since the result of the previous section holds for any derivative that verifies the Black-Scholes equation, we can choose the benchmark securities $C_i(t, K_i)$ as we see fit.
A popular set of benchmark securities, commonly used in the FX market is described next. To simplify notation, we denote $C(K), P(K)$ the call and put of strike $K$, maturity $T$:

- An at-the-money straddle:

$$C_1 = C(S) + P(S)$$

- A "risk reversal", traditionally defined as

$$C_2 = P(K_1) - C(K_2)$$

  with $K_1$ and $K_2$ chosen so that the options have a Delta of .25 in absolute value.

- A "butterfly", defined as

$$C_3 = \beta(P(K_1) + C(K_2)) - (P(S) + C(S))$$

  with $\beta$ determined to set the Vega of the butterfly to 0.

This system is popular because the benchmark securities are very liquid, and because the resulting $A$ matrix of (10.3) is almost diagonal, which allows an intuitive interpretation of the coefficients $x_i$.
To summarize, the calculation steps for pricing an option, taking the smile cost into account, are as follows:

| Strike | Volatility |
|---|---|
| 80 | .32 |
| 100 | .30 |
| 120 | .315 |

Table 10.1: Volatility data

1. compute the risk indicators for the option $O$ to be priced:

$$b = \begin{pmatrix} \frac{\partial O}{\partial \sigma} \\ \frac{\partial^2 O}{\partial \sigma^2} \\ \frac{\partial^2 O}{\partial \sigma \partial S} \end{pmatrix} \tag{10.4}$$

2. compute the A matrix

$$A = \begin{pmatrix} \frac{\partial C_1}{\partial \sigma} & \cdots & \frac{\partial C_3}{\partial \sigma} \\ \frac{\partial^2 C_1}{\partial \sigma^2} & \cdots & \frac{\partial^2 C_3}{\partial \sigma^2} \\ \frac{\partial^2 C_1}{\partial \sigma \partial S} & \cdots & \frac{\partial^2 C_3}{\partial \sigma \partial S} \end{pmatrix} \tag{10.5}$$

3. solve for $x$:

$$b = Ax$$

4. the corrected price for $O$ is:

$$O^M(t,K) = O^{BS}(t,K) + \sum_{i=2}^{3} x_i \left( C_i^M(t) - C_i^{BS}(t) \right) \tag{10.6}$$

where $C_i^M(t)$ is the market price and $C_i^{BS}(t)$ the Black-Scholes price (i.e. with flat volatility).

## 10.3 ILLUSTRATIONS

*Volatility Interpolation*

The simplest use of this method is volality interpolation. Given the ATM volatility and at two other strikes, we want to determine the volatility at an arbitrary strike $K$.
The process is illustrated below, with the market data summarized in Table 10.1 for European options with maturity $T = 1$ year. Interest rate is set to 0 for simplicity.
Volatility data is summarized in Table 10.1.

```
> T <- 1
> Spot <- 100
> r <- 0
> b <- 0
> eps <- 0.001
> sigma <- 0.3
> VolData <- list(c(80, 0.32), c(100, 0.3), c(120, 0.315))
```

Define an array of pricing functions for the three benchmark instruments:

```
> C <- c(function(vol = sigma, spot = Spot) GBSOption(TypeFlag = "c",
     S = spot, X = VolData[[1]][1], Time = T, r = r, b = b, sigma = vol)@price,
     function(vol = sigma, spot = Spot) GBSOption(TypeFlag = "c",
         S = spot, X = VolData[[2]][1], Time = T, r = r, b = b,
         sigma = vol)@price, function(vol = sigma, spot = Spot) GBSOption(TypeFlag = "c",
         S = spot, X = VolData[[3]][1], Time = T, r = r, b = b,
         sigma = vol)@price)
```

Next, define utility functions to compute the risk indicators, all by finite difference:

```
> Vega <- function(f, vol, spot = Spot) (f(vol + eps, spot) - f(vol -
     eps, spot))/(2 * eps)
> Vanna <- function(f, vol, spot = Spot) {
     (Vega(f, vol, spot + 1) - Vega(f, vol, spot - 1))/2
 }
> Volga <- function(f, vol) {
     (Vega(f, vol + eps) - Vega(f, vol - eps))/(eps)
 }
```

Finally, the following function computes the Vanna-Volga adjustment to the Black-Scholes price, and the corresponding implied volatility:

```
> VVVol <- function(X) {
     O <- function(vol = sigma, spot = Spot) GBSOption(TypeFlag = "c",
         S = spot, X = X, Time = T, r = r, b = b, sigma = vol)@price
     TV.BS <- O()
     B.vega <- sapply(1:3, function(i) Vega(C[[i]], sigma))
     B.vanna <- sapply(1:3, function(i) Vanna(C[[i]], sigma))
     B.volga <- sapply(1:3, function(i) Volga(C[[i]], sigma))
     O.vega <- Vega(O, sigma)
     O.vanna <- Vanna(O, sigma)
     O.volga <- Volga(O, sigma)
     B.cost <- sapply(1:3, function(i) C[[i]](VolData[[i]][2]) -
         C[[i]](sigma))
     A <- t(matrix(c(B.vega, B.vanna, B.volga), nrow = 3))
     x <- matrix(c(O.vega, O.vanna, O.volga), nrow = 3)
     w <- solve(A, x)
     CF <- t(w) %*% matrix(B.cost, nrow = 3)
     v <- GBSVolatility(TV.BS + CF, "c", Spot, X, T, r, b, 1e-05)
     v
 }
```

We finally use the vanna-volga interpolating function to construct the interpolated smile curve.

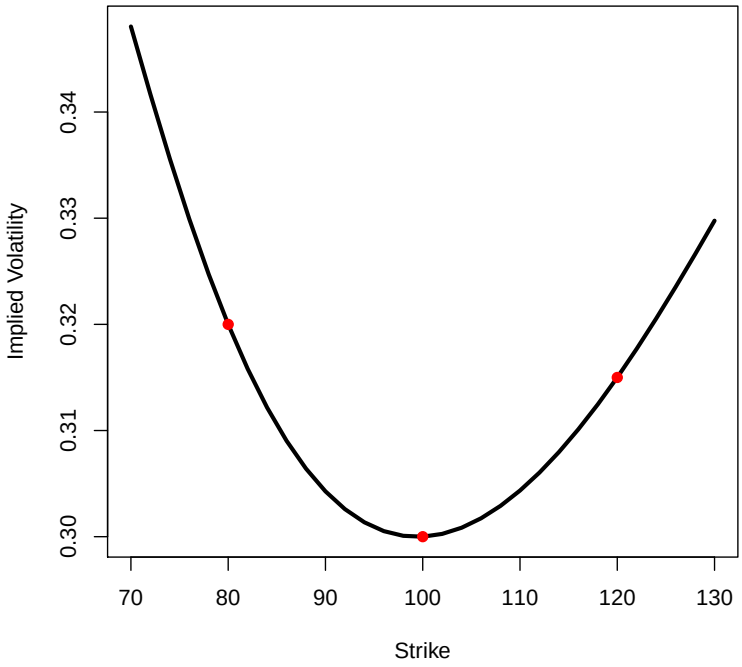



FIGURE 10.1: Interpolated volatility curve with Vanna-Volga algorithm. The 3 benchmark points are shown in red

```
> v <- sapply(seq(70, 130, 2), VVVol)
> plot(seq(70, 130, 2), v, type = "l", lwd = 3, xlab = "Strike",
     ylab = "Implied Volatility")
> points(sapply(VolData, function(v) v[1]), sapply(VolData, function(v) v[2]),
     pch = 19, col = "red")
```

The result is shown in Figure 10.1.

*Pricing a Binary Option*

Consider a one-year binary call, struck at the money. Assume that the smile is quadratic. Again, we assume a null interest rate for simplicity. This time, we use the traditional benchmark instruments of the FX market: straddle, risk-reversal and butterfly, and compute the price of the binary option, adjusted for the smile effect.

```
> T <- 1
```

```
> Spot <- 100
> r <- 0
> d <- 0
> b <- r - d
> sigma <- 30/100
> X <- 110.5
> smile <- function(X) (-(0/20) * (X - Spot) + (1/300) * (X - Spot)^2)/100
```

The strikes corresponding to a 25Δ call and put are computed by inverting the formulae for the Delta of European options. Recall that for a call, the Delta is given by:

$$\Delta = e^{-dT} N(d_1)$$

The strike corresponding to a 25Δ call is therefore:

$$K_{25\Delta} = Se^{-\left(\sigma\sqrt{T}N^{-1}(e^{dT}.25)-(r-d+\frac{\sigma^2}{2})T\right)}$$

```
> alpha <- -qnorm(0.25 * exp(d * T))
> Kp <- Spot * exp(-alpha * sigma * sqrt(T) + (r - d + (1/2) *
      sigma^2) * T)
> Kc <- Spot * exp(alpha * sigma * sqrt(T) + (r - d + (1/2) * sigma^2) *
      T)
```

Define a wrapper function to facilitate calculations on the binary option:

```
> O <- function(vol = sigma, spot = Spot) CashOrNothingOption(TypeFlag = "c",
      S = spot, X = X, K = 100, Time = T, r = r, b = b, sigma = vol)@price
```

The Black-Scholes value, using ATM volatility is:

```
> TV.BS <- O()
> print(paste("BS value:", round(TV.BS, 2)))
[1] "BS value: 31.46"
```

For comparison, we can approximate the binary option with a call spread, giving a value of:

```
> N <- 1000
> TV.CS <- N * (GBSOption("c", Spot, X - 100/(2 * N), T, r, b,
      sigma + smile(X - 100/(2 * N)))@price - GBSOption("c", Spot,
      X + 100/(2 * N), T, r, b, sigma + smile(X + 100/(2 * N)))@price)
> print(paste("Value, approximated by a call spread:", round(TV.BS,
      2)))
[1] "Value, approximated by a call spread: 31.46"
```

We next define the benchmark instruments:

```
> # Put
> P <- function(vol=sigma, spot=Spot) GBSOption(TypeFlag='p', S=spot, X=Kp,
                              Time=T, r=r, b=b, sigma=vol)@price
> # Call
> C <- function(vol=sigma, spot=Spot) GBSOption(TypeFlag='c', S=spot, X=Kc,
                              Time=T, r=r, b=b, sigma=vol)@price
```

```
> # Straddle
> S <- function(vol=sigma, spot=Spot) {
   GBSOption(TypeFlag='c', S=spot, X=Spot, Time=T, r=r, b=b, sigma=vol)@price +
   GBSOption(TypeFlag='p', S=spot, X=Spot, Time=T, r=r, b=b, sigma=vol)@price
 }
> # Risk Reversal
> RR <- function(vol, spot=Spot) {
   P(vol, spot)-C(vol, spot)
 }
> # Butterfly
> BF <- function(vol, spot=Spot, beta=1) {
   beta*(P(vol, spot)+C(vol, spot))-S(vol,spot)
 }
```

The butterfly must be vega-neutral. This is obtained by solving for $\beta$:

```
> BF.V <- function(vol, beta) {
     (BF(vol + eps, beta = beta) - BF(vol - eps, beta = beta))/(2 *
         eps)
 }
> beta <- uniroot(function(b) BF.V(sigma, b), c(1, 1.5))$root
```

Next, we compute the risk indicators for the binary option:

```
> O.vega <- Vega(O, sigma)
> O.vanna <- Vanna(O, sigma)
> O.volga <- Volga(O, sigma)
```

and for the benchmark instruments:

```
> S.vega <- Vega(S, sigma)
> S.vanna <- Vanna(S, sigma)
> S.volga <- Volga(S, sigma)
> RR.vega <- Vega(RR, sigma)
> RR.vanna <- Vanna(RR, sigma)
> RR.volga <- Volga(RR, sigma)
> BF.vega <- 0
> BF.vanna <- Vanna(BF, sigma)
> BF.volga <- Volga(BF, sigma)
```

By definition the smile cost of the straddle is zero, since it is priced with ATM volatility. For the other two benchmark instruments, the smile cost is the difference between the price with the smile effect and the price at the ATM volatility:

```
> RR.cost <- (P(sigma + smile(Kp)) - C(sigma + smile(Kc))) - (P(sigma) -
     C(sigma))
> BF.cost <- beta * (P(sigma + smile(Kp)) + C(sigma + smile(Kc))) -
     beta * (P(sigma) + C(sigma))
```

We can now compute the price correction for the binary option. First the approximate method, ignoring the off-diagonal terms in matrix $A$:

```
> CA <- RR.cost * (O.vanna/RR.vanna) + BF.cost * (O.volga/BF.volga)
```

then the more accurate method, solving the $3 \times 3$ linear system:

```
> A <- matrix(c(S.vega, S.vanna, S.volga, RR.vega, RR.vanna, RR.volga,
     BF.vega, BF.vanna, BF.volga), nrow = 3)
> x <- matrix(c(O.vega, O.vanna, O.volga), nrow = 3)
> w <- solve(A, x)
> CF <- t(w) %*% matrix(c(0, RR.cost, BF.cost), nrow = 3)
```

In summary, we get:

- Black-Scholes price: 31.46
- With approximate Vanna-Volga correction: $31.46 + -4.95 = 26.51$
- With acurate Vanna-Volga correction: $31.46 + -3.5 = 27.96$
- the approximation by a call spread is: 28.79

It is worth noting that a naive calculation, where one would plug the ATM volatility plus smile into the binary option pricing model would yield a very inaccurate result:

```
> P.smile <- O(vol = sigma + smile(X))
```

which yields a value of 31.54. Figure 10.2 compares the values of binary options for a range of strikes, computed with four methods. :

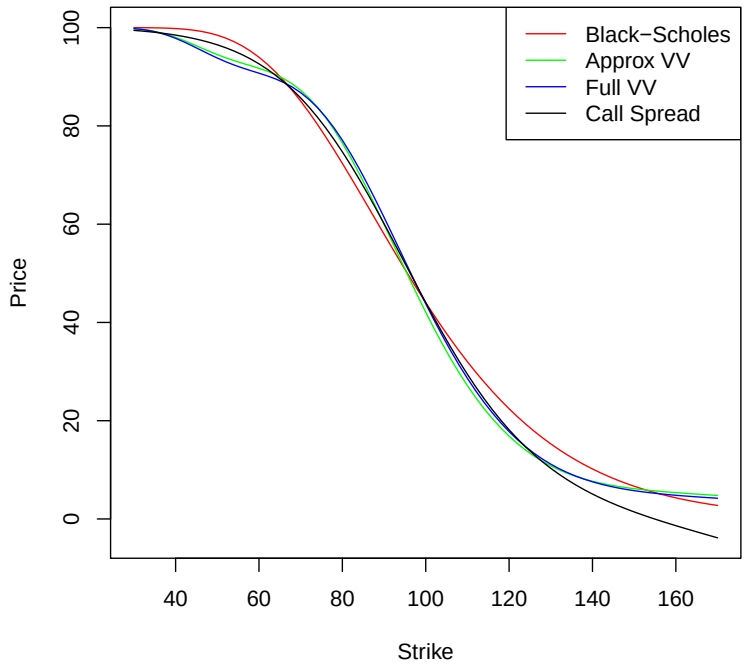



FIGURE 10.2: Price of a binary call as a function of strike, using: ATM volatility (Black-Scholes), diagonal Vanna-Volga correction (Approx VV), exact Vanna-Volga correction (Full VV), and the approximation by a call spread (Call Spread)

# Part V

# Stylized Facts on Asset Returns

# Chapter 11

# Stylized Facts of Financial Data

```
> library(SuppDists)
> library(xtable)
> library(timeSeries)
> library(empfin)
```

In quantitative finance, the term "stylized facts" makes reference to characteristic attributes of market data: time series of prices, term structure of volatility, and the like.
We illustrates the discussion with the time series of a WTI Futures contract traded on the NYMEX. A Futures contract is the obligation to deliver or take delivery of a certain asset at a future date, at a price that is fixed today. In the case of the WTI futures contract, the specifics are as follows:

1. Delivery of 1000 barels of crude oil
2. Oil grade should be "West Texas Intermediate" or equivalent
3. Delivery point is Cushings, a town in Oklahoma which is the convergence point of a large network of pipelines.

We denote $F(t,T)$ the price, observed at time $t$ of a Futures contract for delivery at time $T$. In our example, delivery ($T$) is December 2009. The time series of price is represented in Figure 11.1 and illustrates the speculative commodities bubble of 2008 and the subsequent crash. It is obtained as follows:

```
> data(cl_dec09, package = "empfin")
> dt <- as.Date(cl_dec09[, 1])
> p <- timeSeries(cl_dec09[, 2], dt)
> plot(p, xaxt = "n", ylab = "Price (CL Dec09)", axes = FALSE)
> axis.POSIXct(1, at = seq(dt[1], dt[length(dt)], by = "3 months"),
```

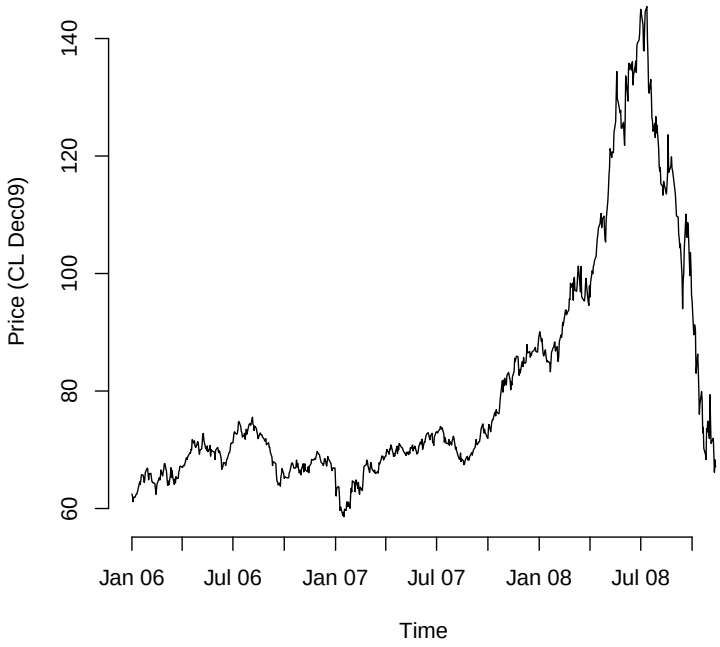



FIGURE 11.1: Daily price of the NYMEX WTI Futures contract for December 2009 delivery.

```
     format = "%b %y")
> axis(2)
```

## 11.1 DISTRIBUTION OF RETURNS

Figure 11.2 shows the time series of daily return, $r_t$, computed by

$$r_t = \ln\left(\frac{F(t+1,T)}{F(t,T)}\right)$$

An histogram of daily returns is shown in Figure 11.3. It is obtained by the command:

```
> r <- r[!is.na(r)]
> hist(r, breaks = 40, freq = FALSE, main = "")
> rug(r)
> curve(dnorm(x, mean = mean(r), sd = sd(r)), add = TRUE, col = "red",
     lwd = 3, xaxt = "n")
```

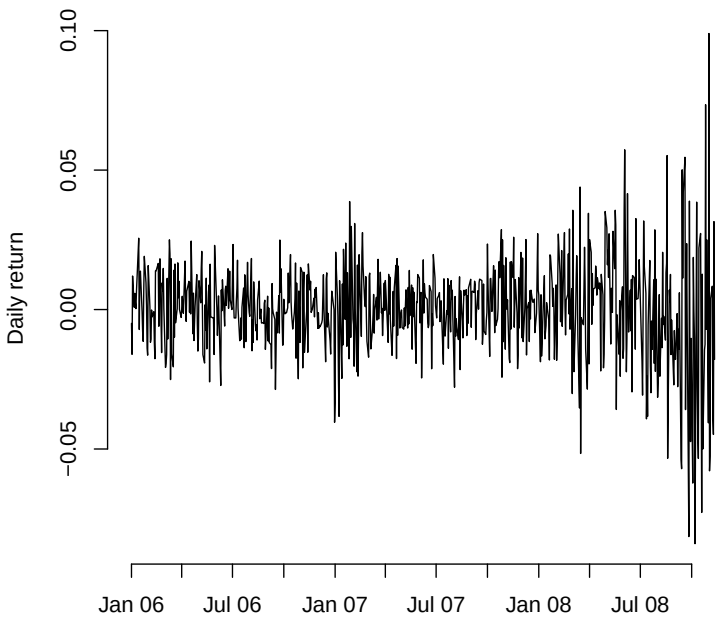



FIGURE 11.2: Daily return of WTI Futures contract for December 2009 delivery

To start the analysis, we plot the sample cumulative distribution against the theoretical normal cumulative distribution with same mean and standard deviation:

```
> qqnorm(as.vector(r), main = NULL)
> qqline(as.vector(r), col = "red", lwd = 2)
```

The non-normality of daily return is apparent in Figure 11.4.

The first 4 sample moments are summarized in Table 11.1. Notice that the value for the 4th moment is the excess kurtosis, i.e. the actual kurtosis - 3.

```
> mo <- list(mean = mean(r), stdev = sqrt(var(r)), skewness = skewness(r),
      kurtosis = kurtosis(r))
```

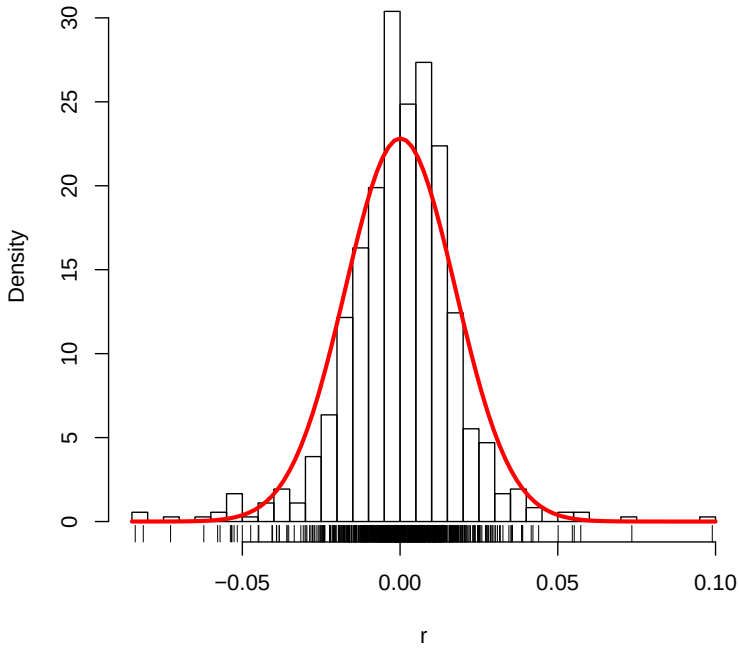



FIGURE 11.3: Histogram of daily returns, WTI Futures for December 2009 delivery, and normal density with same mean and standard deviation.

| | |
|---|---|
| mean | 0.0001 |
| stdev | 0.0175 |
| skewness | -0.2867 |
| kurtosis | 3.9711 |

Table 11.1: Sample moments of daily return, WTI Futures contract for December 2009 delivery

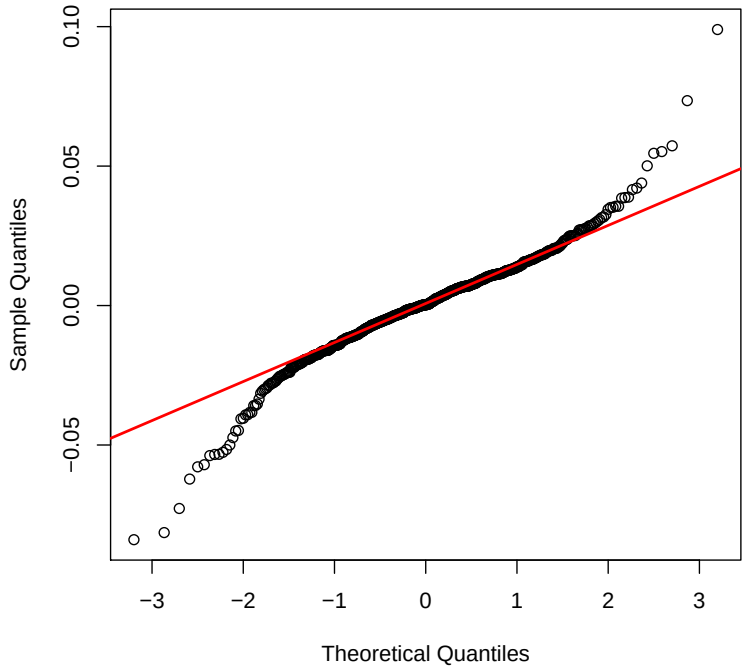



FIGURE 11.4: QQ plot of daily returns

## 11.2 FITTING A DENSITY TO HISTORICAL RETURNS

Given the significant values for the skew and excess kurtosis in sample data, we look for 4-parameter densities. There are many such distributions, and select, among many alternatives, the Johnson family of distributions. The Johnson family of distributions is formed by various transformations of the normal density. Let $X$ be the observed data, an define $Z$ by:

$$Z = \gamma + \delta \ln\left(g\left(\frac{x-\xi}{\lambda}\right)\right) \tag{11.1}$$

where:

$$g(u) = \begin{cases} u & SL \\ u + \sqrt{1+u^2} & SU \\ \frac{u}{1-u} & SB \\ e^u & SN \end{cases}$$

$X$ follows a Johnson distribution if $Z$ is normal.

| | gamma | delta | xi | lambda | type |
|---|---|---|---|---|---|
| 1 | 0.1498 | 1.5579 | 0.0023 | 0.0205 | SU |

Table 11.2: Fitted Johnson SU parameters

| | sample | johnson |
|---|---|---|
| mean | 9.86e-05 | -1.24e-04 |
| sigma | 1.75e-02 | 1.65e-02 |
| skew | -2.87e-01 | -2.82e-01 |
| kurt | 3.99e+00 | 3.50e+00 |

Table 11.3: Sample and fitted moments with Johnson density

We fit the SU distribution with the command:

```
> johnson.fit <- JohnsonFit(r)
```

The fitted coefficients of the Johnson SU distribution are summarized in table 11.2, with
The empirical and fitted moments are summarized in Table 11.3. We finally use the Kolmogorov-Smirnov test to check if the sample follow the fitted Johnson distribution.
Figure **??** overlays the fitted Johnson SU distribution to the sample histogram.

```
> pp3 <- function(x) format(x, scientific = TRUE, digits = 3)
> t2 = substitute(paste(gamma, ": ", g, " ", delta, ": ", d, " ",
     xi, ": ", x, " ", lambda, ": ", l), list(g = pp3(johnson.fit$gamma),
     d = pp3(johnson.fit$delta), x = pp3(johnson.fit$xi), l = pp3(johnson.fit$lambda)))
> hist(r, freq = FALSE, breaks = 40, main = t2)
> plot(function(x) dJohnson(x, johnson.fit), -0.1, 0.1, col = "red",
     lwd = 3, add = TRUE)
```

The flexibility of the Johnson distribution, and its ability to match the heavy tails of the sample distribution is visible when one compares the normal QQ plot to the fitted Johnson SU QQ plot in Figure **??**.
This is achieved with the following code:

```
> qqJohnson <- function(y, params, n = 100, abline = TRUE, ...) {
     u <- seq(from = 1/(n + 1), by = 1/(n + 1), length = n)
     q <- qJohnson(u, params)
     if (abline) {
         ret <- qqplot(q, y, ...)
         abline(0, 1)
     }
     else {
         ret <- qqplot(q, y, ...)
     }
     invisible(ret)
```

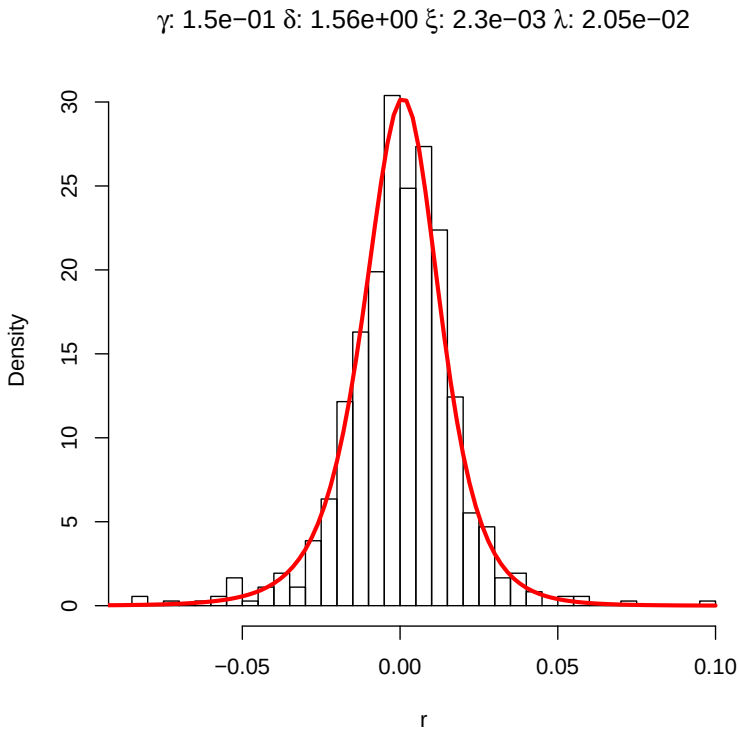



FIGURE 11.5: Histogram of daily return and fitted Johnson SU distribution

```
}
> op <- par(mfrow = c(1, 2))
> qqJohnson(r, johnson.fit, main = "Johnson SU")
> qqnorm(r, main = "Normal")
> qqline(r)
> par(op)
```

## 11.3 AUTOCORRELATION OF RETURNS

We now turn to the autocorrelation structure of the return series. The autocorrelograms of return and of the absolute value of return are shown in fig **??**. There is no evidence of autocorrelation in the return series. However the absolute return shows a clear first-order autocorrelation pattern: days with large returns (in absolute value) tend to be followed by other days with large mouvements. The same holds true for days with small returns. This feature is named “volatility clustering”.

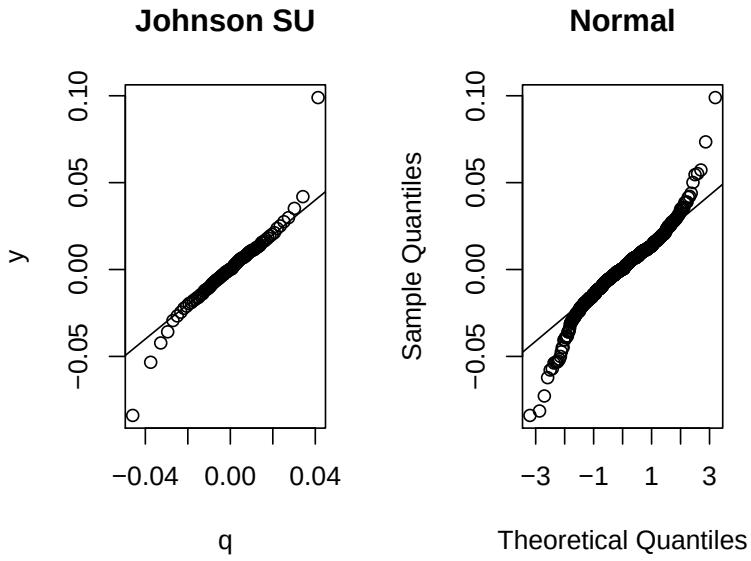
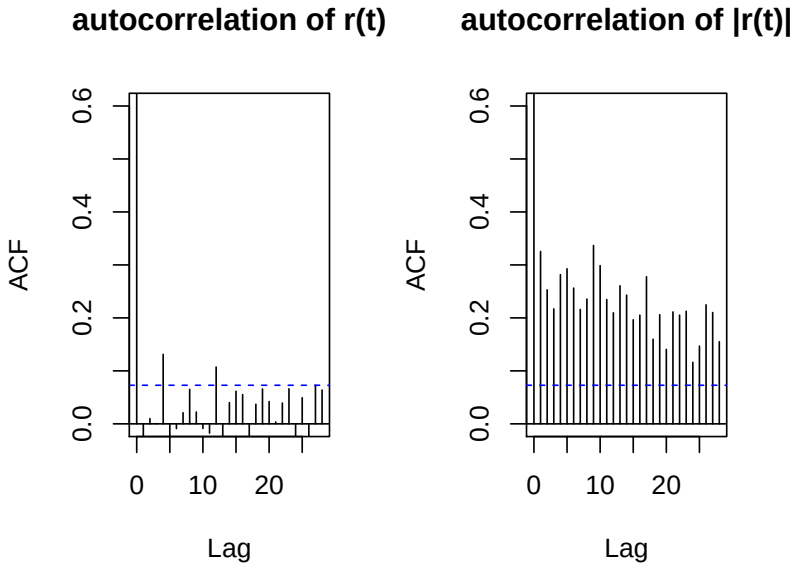



FIGURE 11.6: Johnson vs. Normal distribution fitted to daily WTI Futures returns

```
> op <- par(mfrow = c(1, 2))
> acf(r, ylim = c(0, 0.6), main = "autocorrelation of r(t)")
> acf(abs(r), ylim = c(0, 0.6), main = "autocorrelation of |r(t)|")
> par(op)
```

To summarize, empirical observations show that the distribution of returns exhibit features that strongly repart from the classical hypothesis of independence and normality. We find:

1. no evidence of linear autocorrelation of return, however,
2. there is an observable autocorrelation of $|r_t|$ and $r_t^2$, suggesting autocorrelation in the volatility of return;
3. we also observe large excess kurtosis, which is incompatible with normal density, finally we find that
4. the distribution of returns is well approximated by a Johnson SU distribution.


autocorrelation of r(t)
autocorrelation of |r(t)|
ACF
0.0
0.2
0.4
0.6
0
10
20
Lag
ACF
0.0
0.2
0.4
0.6
0
10
20
Lag

# Part VI

# The Volatility Surface

# CHAPTER 12

# THE IMPLIED VOLATILITY

Listed options are quoted in price. The implied volatility is the volatility that recovers that quoted price, using the Black-Scholes model. Computing implied volatility amounts to inverting the Black-Scholes formula, given an option price. This chapter discusses the issues involved in such calculation, and how to represent the result.

## 12.1 MARKET DATA

Option settlement prices are published every day by exchanges. Figure 12.1 shows a sample of the data file published every day by the NYMEX (CME Group). A csv file containing daily option settlement quotes is available for download (as of October 2012) at http://www.cmegroup.com/market-data/settlements.

LISTING 12.1: OPTION SETTLEMENT PRICES. SOURCE: NYMEX

```
                NEW YORK MERCANTILE EXCHANGE
          NYMEX OPTIONS CONTRACT LISTING FOR 12/29/2008

                              TODAY'S  PREVIOUS  ESTIMATED
--------CONTRACT--------      SETTLE    SETTLE     VOLUME
LC    02 09    P   30.00        .53       .85          0
LC    02 09    P   35.00       1.58      2.28          0
LC    02 09    P   37.50       2.44      3.45          0
LC    02 09    C   40.00       3.65      2.61         10
LC    02 09    P   40.00       3.63      4.90          0
LC    02 09    P   42.00       4.78      6.23          0
LC    02 09    C   42.50       2.61      1.80          0
LC    02 09    C   43.00       2.43      1.66          0
LC    02 09    P   43.00       5.41      6.95        100
```

Notice that settlement prices are provided even for options that have not been traded today. Providing a price for all strikes and maturities is necessary in order to compute margin calls on open positions. The exchange uses an interpolation method for estimating these settlement prices in the absence of actual transactions.

## 12.2 CALCULATION OF IMPLIED VOLATILITY

Given an observed call price $C^*$, we look for the volatility $\sigma$ such that:

$$\begin{aligned} C^* &= C_{\text{BS}}(S, K, T, r, \sigma) \\ &= C(\sigma) \end{aligned} \tag{12.1}$$

where $C_{\text{BS}}(S, K, T, r, \sigma)$ is the Black-Scholes formula for the price of a call (8.1), with:

$S$ Spot price

$K$ Strike

$T$ Time to expiry

$r$ Risk-free rate

$\sigma$ Volatility

In theory, because of the call-put parity relationship, the volatility associated with a strike $K$ can be equivalently computed from the call price or the put price. Any discrepancy beyond transaction cost would create an arbitrage opportunity.

### *Newton's method*

Newton's algorithm is a popular method for solving (12.1). However, one must carefully choose the initial value of the iteration in order to ensure convergence.
Observe that the Black-Scholes price is an increasing function of volatility:

$$\frac{\partial C}{\partial \sigma} = Sn(d_1)\sqrt{T}$$

with

$$\begin{aligned} \lim_{\sigma \to 0} C(\sigma) &= (S_t - Ke^{-rT})^+ \\ \lim_{\sigma \to \infty} C(\sigma) &= S_t \end{aligned}$$

Moreover, there is a change in convexity in $C(\sigma)$:

$$\frac{\partial^2 C}{\partial \sigma^2} = S\sqrt{T}n(d_1)\frac{1}{\sigma}\left[\frac{1}{\sigma^2 T}\ln\left(\frac{F}{K}\right)^2 - \frac{1}{4}\sigma^2 T\right]$$

with $F = Se^{rT}$.
Thus, $C(\sigma)$ is convex on the interval $(0, \sqrt{\frac{2|\ln(F/K)|}{T}}]$, and concave otherwise.
This implies that the equation $C(\sigma) = C^*$ for all

$$(S_t - K)^+ < C < S_t$$

can be solved by Newton's method, starting at

$$\sigma_0 = \sqrt{\frac{2|\ln(F/K)|}{T}}$$

since the iteration

$$\sigma_{n+1} = \sigma_n + \frac{C^* - C(\sigma_n)}{\frac{\partial C}{\partial \sigma}}$$

generates a monotonous sequence. The algorithms may be summarized as follows:

1. Set $\sigma_0 = \sqrt{\frac{2|\ln(F/K)|}{T}}$
2. While $|C(\sigma_n) - C^*| > \epsilon$:
   a) Let
   $$\sigma_{n+1} = \sigma_n + \frac{C^* - C(\sigma_n)}{\frac{\partial C}{\partial \sigma}}$$
   b) $n \leftarrow n + 1$

The implementation is straight-forward:

```
> ImpliedVolNewton <- function(p, TypeFlag, S, X, Time, r, b, tol,
    maxiter = 50) {
    F <- S * exp((r - b) * T)
    s <- sqrt(2 * abs(log(F/X))/T)
    not_converged <- T
    vega <- GBSGreeks(Selection = "vega", TypeFlag, S, X, Time,
        r, b, s)
    i <- 1
    while (not_converged & (i < maxiter)) {
        err <- (p - GBSOption(TypeFlag, S, X, Time, r, b, s)@price)
        s <- s + err/vega
        not_converged <- (abs(err/vega) > tol)
        i <- i + 1
    }
    s
}
```

| | Default | Modified |
|---|---|---|
| user.self | 0.58 | 0.20 |
| sys.self | 0.00 | 0.00 |
| elapsed | 0.58 | 0.20 |

Table 12.1: Implied volatility timing test: default calculation and Newton's method started at inflexion point

To test this strategy, we perform a timing test by executing 100 replications of the same calculation, and collecting statistics for the default implementation and for Newton's method started at the inflexion point. The results are summarized in Table 12.1, and demonstrate the advantage of choosing the initial point described above.

```
> TypeFlag <- "c"
> S <- 100
> X <- 110
> Time <- 1
> r <- 0.03
> b <- 0.01
> sigma <- 0.314
> tol <- 1e-06
> p <- GBSOption(TypeFlag, S, X, Time, r, b, sigma)@price
> t1 <- function(n = 100) {
      for (i in 1:n) {
          si <- GBSVolatility(p, TypeFlag, S, X, Time, r, b, tol = tol,
              maxiter = 50)
      }
      si
  }
> t2 <- function(n = 100) {
      for (i in 1:n) {
          si <- ImpliedVolNewton(p, TypeFlag, S, X, Time, r, b,
              tol)
      }
      si
  }
> stats.t1 <- system.time(t1(100))
> stats.t2 <- system.time(t2(100))
> xm <- xtable(data.frame(cbind(Default = stats.t1, Modified = stats.t2))[1:3,
      ])
```

*Derivative-free Methods*

For deep in the money or out of the money options, where Vega is very small, a solution method that does not require a derivative may be preferred. This class of algorithms include the bisection, secant and *regula falsi* methods. These algorithms share a common logic: they start with a

bracketing interval around the solution, and progressively shrinks it. As an illustration, we test here the bisection algorithms:

```
> ImpliedVolBisection <- function(p, TypeFlag, S, X, Time, r, b,
     tol, sBounds, maxiter = 100) {
     sMin <- min(sBounds)
     sMax <- max(sBounds)
     pMin <- GBSOption(TypeFlag, S, X, Time, r, b, sMin)@price
     pMax <- GBSOption(TypeFlag, S, X, Time, r, b, sMax)@price
     not_converged <- abs(pMin - pMax) > tol
     i <- 1
     while (not_converged & (i < maxiter)) {
         sStar <- (sMin + sMax)/2
         pStar <- GBSOption(TypeFlag, S, X, Time, r, b, sStar)@price
         if (pStar < p) {
             pMin <- pStar
             sMin <- sStar
         }
         else {
             pMax <- pStar
             sMax <- sStar
         }
         not_converged <- (abs(pMin - pMax) > tol)
         i <- i + 1
     }
     (sMin + sMax)/2
 }
```

A timing test summarized in Table 12.2 shows the value of this algorithm for a deep in the money call.

```
> t3 <- function(n = 100) {
     for (i in 1:n) {
         si <- ImpliedVolBisection(p, TypeFlag, S, X, Time, r,
             b, tol, c(0.2, 0.7))
     }
     si
 }
```

```
> S <- 100
> X <- 20
> sigma <- 0.514
> p <- GBSOption(TypeFlag, S, X, Time, r, b, sigma)@price
> n <- 100
> stats.t21 <- system.time(t1(n))
> stats.t22 <- system.time(t2(n))
> stats.t23 <- system.time(t3(n))
> xm <- xtable(data.frame(cbind(Default = stats.t21, ModifiedNewton = stats.t22,
     Bisection = stats.t23))[1:3, ])
```

| | Default | ModifiedNewton | Bisection |
|---|---|---|---|
| user.self | 1.01 | 2.33 | 0.86 |
| sys.self | 0.01 | 0.01 | 0.00 |
| elapsed | 1.02 | 2.35 | 0.86 |

Table 12.2: Implied volatility timing test: default calculation, modified Newton and bisection methods

*Jaeckel's method*

Jackel (2006) proposes various modifications to regularize Newton's method. Rewriting the Black-Scholes formula, the implied volatility calculation amounts to solving the following equation for $\sigma$

$$p = \delta\theta\left[F\Phi\left(\theta\left[\frac{\ln(F/K)}{\sigma}+\frac{\sigma}{2}\right]\right)-K\Phi\left(\theta\left[\frac{\ln(F/K)}{\sigma}-\frac{\sigma}{2}\right]\right)\right]$$

where:

$\delta$ discount factor

$\theta$ 1 for call, -1 for put

$F$ Forward price: $Se^{(r-d)T}$

$\sigma$ $\sigma\sqrt{T}$

Furthermore, define the normalized price $b$ as:

$$\begin{aligned} x &= \ln(F/K) \\ b &= \frac{p}{\delta\sqrt{FK}} \end{aligned}$$

The Black-Scholes equation becomes:

$$b = \theta\left[e^{x/2}\Phi\left(\theta\left[\frac{x}{\sigma}+\frac{\sigma}{2}\right]\right)-e^{-x/2}\Phi\left(\theta\left[\frac{x}{\sigma}-\frac{\sigma}{2}\right]\right)\right] \tag{12.2}$$

which has the advantage of reducing the complexity of the function evaluation. This has a substantial impact on performance. To see this, we apply Newton's method to (12.2).
First, define a function that computes the normalized Black-Scholes price:

```
> NormBSPrice <- function(TypeFlag, S, K, Time, r, b) {
    d = r - b
    F = S * exp((r - d) * Time)
    x = log(F/K)
    ex = exp(x/2)
    ex.inv = 1/ex
```

```
        theta = ifelse(TypeFlag == "c", 1, -1)
        foo <- function(sigma) {
            s = sigma * sqrt(Time)
            xs = x/sigma
            s2 = sigma/2
            theta * (ex * pnorm(theta * (xs + s2)) - ex.inv * pnorm(theta *
                (xs - s2)))
        }
        foo
    }
```

and use that function in the modified Newton's method:

```
> ImpliedVolNewton2 <- function(p, TypeFlag, S, X, Time, r, b,
    tol, maxiter = 50) {
    F <- S * exp(b * Time)
    s <- sqrt(2 * abs(log(F/X))/T)
    p.bar <- p/(exp(-r * Time) * sqrt(F * X))
    pBar <- NormBSPrice(TypeFlag, S, X, Time, r, b)
    not_converged <- T
    vega <- GBSGreeks(Selection = "vega", TypeFlag, S, X, Time,
        r, b, s)/(exp(-r * Time) * sqrt(F * X))
    i <- 1
    while (not_converged & (i < maxiter)) {
        err <- (p.bar - pBar(s))
        s <- s + err/vega
        not_converged <- (abs(err/vega) > tol)
        i <- i + 1
    }
    s
}
```

This modified implementation is finally tested with:

```
> TypeFlag <- "c"
> S <- 100
> X <- 110
> Time <- 1
> r <- 0.03
> b <- 0.01
> sigma <- 0.314
> tol <- 1e-06
> p <- GBSOption(TypeFlag, S, X, Time, r, b, sigma)@price
> t4 <- function(n = 100) {
    for (i in seq(n)) si <- ImpliedVolNewton2(p, TypeFlag, S,
        X, Time, r, b, tol)
    si
}
> stats.t4 <- system.time(t4(100))
> xm <- xtable(data.frame(cbind(Default = stats.t1, Modified = stats.t2,
    Jaeckel = stats.t4))[1:3, ])
```

The results are summarized in Table 12.3, and show the substantial improvement provided by this simple refactoring of the Black-Scholes equation.

| | Default | Modified | Jaeckel |
|---|---|---|---|
| user.self | 0.58 | 0.20 | 0.01 |
| sys.self | 0.00 | 0.00 | 0.00 |
| elapsed | 0.58 | 0.20 | 0.01 |

Table 12.3: Implied volatility timing test: default calculation, modified Newton's and modified Newton's with Jaeckel's parametrization.

Jaeckel defines a further transformation to avoid the inflexion point identified earlier, but that does not seem to speed up the solution any further. The calculation of the implied volatility for a given strike and maturity is the first step towards building a volatility surface, which is described in the next chapter.

# CHAPTER 13

# BUILDING AN EQUITY VOLATILTY SURFACE

The purpose of this chapter is to present a step-by-step procedure for building a volatily surface, using the S&P 500 index options for illustration.

## 13.1 THE DATA

Settlement prices for call and put options on the S&P 500 index are available for download at http://www.cboe.com/DelayedQuote/QuoteTableDownload.aspx. fragment of the data file is shown in Table 13.1.
The ticker for S&P 500 index options (i.e. (SPXW1128A1075-E)) should be interpreted as follows:

- Characters 5-9: the expiry date of the option, and the option type (call or put):
  - Character 5-6: year
  - Character 7-8: day of month
  - Character 9: month, coded differently for calls and puts. For calls: A (January) to L (December), for puts: M (January) to X (December).
- Character 10-13: option strike.

For example, (SPXW1128A1075-E) is the ticker of a European call, strike 1075, expiring on January 28, 2011.
The ticker is parsed with the following function:

```
> SPX_months <- hash(keys = unlist(strsplit("ABCDEFGHIJKLMNOPQRSTUVWX",
     split = "")), values = c(1:12, 1:12))
> spx_symbol = "\\(SPX(1[0-9])([0-9]{2})([A-Z])([0-9]{3,4})-E\\)"
> parse_SPX_expiry <- function(tokens) {
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SPX (SP 500 INDEX) | 1290.59 | +7.24 | | | | | |
| Jan 24 2011 @ 14:03 ET | | | | | | | |
| Calls | Last Sale | Net | Bid | Ask | Vol | Open Int | ... |
| 11 Jan 1075.00 (SPXW1128A1075-E) | 0.0 | 0.0 | 215.30 | 217.00 | 0 | 0 | ... |
| 11 Jan 1100.00 (SPXW1128A1100-E) | 0.0 | 0.0 | 190.60 | 191.80 | 0 | 0 | ... |

Table 13.1: Fragment of S&P500 Option settlement data downloaded from CBOE web site.Data for calls are shown here. Data for puts follow in the next 7 columns, with the same structure.

```
    year = 2000 + as.integer(tokens[2])
    day = as.integer(tokens[3])
    month = SPX_months[[tokens[4]]]
    myDate(sprintf("%02d-%02d-%04d", day, month, year))
}
```

The next function reads the csv file and builds a data frame of call and put prices:

```
> read_SPX_file <- function(option_data_file) {
    df = read.csv(option_data_file, sep = ",", header = F, nrow = 2)
    dtTradeTokens = unlist(strsplit(as.character(df[2, "V1"]),
        " ", fixed = T)[1:3])
    dtTrade = myDate(paste(dtTradeTokens[c(2, 1, 3)], collapse = ""))
    spot = df[1, "V2"]
    df = read.csv(option_data_file, sep = ",", header = T, skip = 2)
    call_df = df[, c("Calls", "Bid", "Ask")]
    colnames(call_df) = c("Spec", "PBid", "PAsk")
    call_df["CP"] = "C"
    put_df = df[, c("Puts", "Bid.1", "Ask.1")]
    colnames(put_df) = c("Spec", "PBid", "PAsk")
    put_df["CP"] = "P"
    df_all = rbind(call_df, put_df)
    df_all$Type = "Euro"
    list(spot = spot, dtTrade = dtTrade, quotes = df <- all)
}
```

Finally, the following script filters out bad data, and creates a data frame with columns:

- dtTrade: Quote date, or time stamp
- Strike
- dtExpiry: Option expiry date
- CP: Call/Put flag, coded as C/P or Call/Put
- Spot: Price of underlying asset
- Type: European/American

- PBid: Bid price
- PAsk: Ask price

```
option_data_file = <path to csv file>

res = read_SPX_file(option_data_file)

df_all <- res$df_all

# parse Spec column for strike and expiry date
# only retain valid data
tokens <- str_match(df_all[,'Spec'], spx_symbol)

good_rows <- apply(tokens, 1, function(x) all(!is.na(x)))

df_all <- df_all[good_rows,]
tokens <- tokens[good_rows,]
df_all['Strike'] = as.numeric(tokens[,5])

dtExpiry = apply(tokens, 1, parse_SPX_expiry)
df_all['dtExpiry'] = as.Date(dtExpiry, origin='1970-01-01')

df_all = df_all[(df_all['Strike'] > 0) & (df_all['PBid']>0)
                    & (df_all['PAsk']>0),]

df_all['dtTrade'] = res$dtTrade
df_all['Spot'] = res$spot

print(paste('Nb of records processed:', length(df_all$PBid)))
save(df_all, file='df_SPX.rda')
write.csv(df_all, 'df_SPX.csv')
```

## 13.2 ESTIMATION OF THE VOLATILITY SURFACE

**Implied risk-free rate and dividend yield** Rather than using exogenous information, we will estimate the risk-free rate and dividend yield implied by the option market itself.
Recall the put-call parity relationship for European options with continuous dividends:

$$C_t - P_t = S_t e^{-d(T-t)} - K e^{-r(T-t)}$$

where

$C_t$ price of call at time $t$

$P_t$ price of put at time $t$

$S_t$ spot price of underlying asset

$d$ continuous dividend yield

$r$ risk-free rate

$T$ Expity

Because of measurements errors and bid-ask spreads, this relationship does not hold exactly, however, for each maturity, we estimate the terms $e^{-d(T-t)}$ and $e^{-r(T-t)}$ by estimating the linear regression:

$$C_t - P_t = a_0 + a_1 K$$

which yields the following estimates for the risk-free rate and dividend yield of maturity $T$.

$$\begin{aligned} r &= -\frac{1}{T}\ln(-a_1) \\ d &= \frac{1}{T}\ln\left(\frac{S_t}{a_0}\right) \end{aligned}$$

**Forward at-the-money volatility** The next step is to estimate the implied volatility of the an option struck at the forward price. In general, such option is not traded, and the volatility must therefore be estimated. The calculation involves 3 steps, performed separately on calls and puts:

1. Estimate the bid ($\sigma_b(K)$) and ask ($\sigma_a(K)$) Black-Scholes volatility for each strike $K$.
2. Compute a mid-market implied volatility for each strike:
$$\sigma(K) = \frac{\sigma_b(K) + \sigma_a(K)}{2}$$
3. Let $F$ be the forward price, the corresponding mid-market implied volatility is computed by linear interpolation between the two strikes braketing $F$.
4. The forward ATM volatility is the average of the volatilities computed on calls and puts.

**Quick-delta** The "Quick Delta" (QD) is a popular measure of moneyness, inspired by the definition of the Delta of a European call:

$$QD(K) = N\left(\frac{1}{\sigma\sqrt{T}}\ln\left(\frac{F_T}{K}\right)\right)$$

Note that $QD(F_T) = 0.5$, for all maturities, while the regular forward Delta is a function of time to expiry. This property of "Quick Delta" makes it

easy to interpret. In addition, being bounded by 0 and 1, it is a convenient measure of moneyness for representing the volatility smile.

The function `compute.iv()`, reproduced below, performs these calculations for one expiry date.

```
> compute.iv <- function(group, tMin = 0, nMin = 0, QDMin = 0,
    QDMax = 1, keepOTMData = TRUE) {
    df.out = data.frame()
    dtTrade = group$dtTrade[1]
    dtExpiry = group$dtExpiry[1]
    spot = group$Spot[1]
    daysToExpiry = as.numeric(dtExpiry - dtTrade)
    timeToMaturity = daysToExpiry/365
    if (timeToMaturity < tMin)
        return(df.out)
    df_call = subset(group, Type == "C", select = c(Strike, PBid,
        PAsk))
    df_put = subset(group, Type == "P", select = c(Strike, PBid,
        PAsk))
    if ((nrow(df_call) < nMin) | (nrow(df_put) < nMin))
        return(df.out)
    df_call$PremiumC = (df_call$PBid + df_call$PAsk)/2
    df_put$PremiumP = (df_put$PBid + df_put$PAsk)/2
    df_all = merge(df_call, df_put, by = "Strike", all = TRUE)
    df_all$CP = df_all$PremiumC - df_all$PremiumP
    model = lm(CP ~ Strike, df_all)
    a0 = model$coefficients[1]
    a1 = model$coefficients[2]
    iRate = -log(-a1)/timeToMaturity
    dRate = log(spot/a0)/timeToMaturity
    discountFactor = exp(-iRate * timeToMaturity)
    Fwd = spot * exp((iRate - dRate) * timeToMaturity)
    print(paste("Fwd: ", round(Fwd, 2), " int rate: ", round(iRate *
        100, 2), "div yield: ", round(dRate * 100, 2)))
    impvol <- function(aRow, cp, p.column) {
        res = try(GBSVolatility(price = aRow[p.column], TypeFlag = cp,
            S = spot, X = aRow["Strike"], Time = timeToMaturity,
            r = iRate, b = iRate - dRate))
        if (!is.numeric(res))
            res = NA
        res
    }
    df_call$IVBid <- apply(df_call, 1, FUN = impvol, cp = "c",
        p.column = "PBid")
    df_call$IVAsk = apply(df_call, 1, FUN = impvol, cp = "c",
        p.column = "PAsk")
    df_call$IVMid <- (df_call$IVBid + df_call$IVAsk)/2
    df_put$IVBid = apply(df_put, 1, FUN = impvol, cp = "p", p.column = "PBid")
    df_put$IVAsk = apply(df_put, 1, FUN = impvol, cp = "p", p.column = "PAsk")
    df_put$IVMid <- (df_put$IVBid + df_put$IVAsk)/2
    atm.vol.c = approx(df_call$Strike, df_call$IVMid, xout = Fwd)$y
    atm.vol.p = approx(df_put$Strike, df_put$IVMid, xout = Fwd)$y
    atmVol = (atm.vol.c + atm.vol.p)/2
    print(paste("ATM vol: ", atmVol))
```

```
        df_call$QuickDelta = pnorm(log(Fwd/df_call$Strike)/(atmVol *
            sqrt(timeToMaturity)))
        df_put$QuickDelta = pnorm(log(Fwd/df_put$Strike)/(atmVol *
            sqrt(timeToMaturity)))
        df_call = df_call[(df_call$QuickDelta >= QDMin) & (df_call$QuickDelta <=
            QDMax), ]
        df_put = df_put[(df_put["QuickDelta"] >= QDMin) & (df_put["QuickDelta"] <=
            QDMax), ]
        df_call$PremiumC <- NULL
        df_call$Type <- "C"
        df_put$PremiumP <- NULL
        df_put$Type <- "P"
        df_cp = rbind(df_call, df_put)
        df_cp$iRate = iRate
        df_cp$iDiv = dRate
        df_cp$ATMVol = atmVol
        df_cp$Fwd = Fwd
        df_cp$dtTrade = dtTrade
        df_cp$dtExpiry = dtExpiry
        df_cp
    }
```

The function must be called for each expiry date. The implied volatility surface is build one expiry date at a time, as follows:

```
  load(file = 'df_SPX.rda')
  SPX.IV = data.frame()
  for(dt in unique(df_SPX$dtExpiry)) {
  dt.2 <- as.Date(dt, origin='1970-01-01')
  print(paste('DtExpiry:', dt.2))
  tmp = subset(df_SPX, dtExpiry == dt.2)
  df.tmp = compute.iv(tmp, tMin=1/12, nMin=6, QDMin=.2, QDMax=.8,
       keepOTMData=TRUE)
  SPX.IV = rbind(SPX.IV, df.tmp)
}
```

The resulting data set may be found in package `empfin`. The data is represented in Figure 13.1.

```
    > data(SPX.IV, package = "empfin")
    > df_call <- subset(SPX.IV, Type == "C")
    > x <- df_call$QuickDelta
    > y <- as.numeric(df_call$dtExpiry)
    > z <- df_call$IVMid
    > s <- interp(x, y, z)
    > nrz = nrow(s$z)
    > ncz = ncol(s$z)
    > jet.colors = colorRampPalette(c("blue", "green"))
    > nbcol = 100
    > color = jet.colors(nbcol)
    > zfacet = s$z[-1, -1] + s$z[-1, -ncz] + s$z[-nrz, -1] + s$z[-nrz,
        -ncz]
    > facetcol = cut(zfacet, nbcol)
    > TTM <- as.numeric(as.Date(s$y, origin = "1970-01-01") - df_call$dtTrade[1])/365
    > persp(s$x, TTM, s$z * 100, col = color[facetcol], xlab = "\nQuick Delta",
```

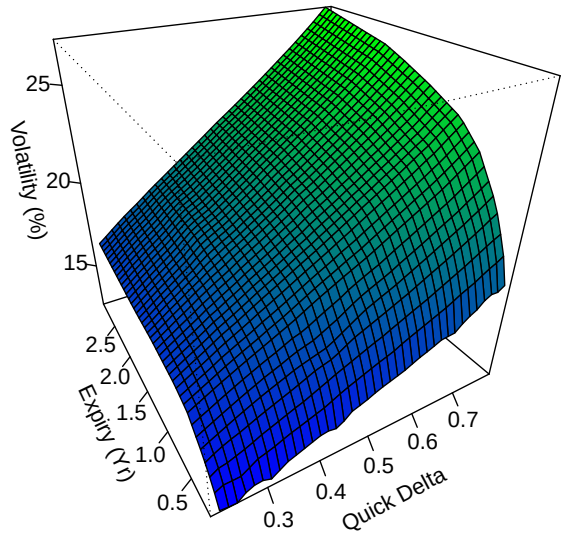



FIGURE 13.1: Imnplied volatility surface, SPX index options, 24-Jan-2011

```
ylab = "\nExpiry (Yr)", zlab = "\nVolatility (%)", theta = -30,
phi = 30, ticktype = "detailed")
```

# CHAPTER 14

# THE IMPLIED DISTRIBUTION OF ASSET PRICE

```
> library(fOptions)
> library(fExoticOptions)
> library(fAsianOptions)
> library(fInstrument)
> library(SuppDists)
> library(empfin)
> data(CL.ImpliedVol)
```

In this chapter, we consider models that account for the observed "volatility smile". We start by extracting the distribution of returns implied by option prices, using a procedure due to Breeden & Litzenberger (1978). With a analytical expression for the volatility smile, one can use the Breeden-Litzenberger formula to derive a closed-form expression for the corresponding density function. This will be illustrated with a simple quadratic model of volatility, due to Shimko (1993). Finally, we construct trinomial trees that are consistent with the observed volatility smile.

## 14.1 THE BREEDEN-LITZENBERGER FORMULA

The Breeden-Litzenberger formula gives the risk-neutral density of the underlying asset at maturity $T$ as a function of derivative prices:

$$p_T(K) = e^{rT} \frac{\partial^2 C(S,K,T)}{\partial K^2} \tag{14.1}$$

where $C(S,K,T)$ is the price of a call of strike $K$, maturity $T$, when the current spot is $S$.
By definition of the risk-neutral probability,

$$C(S,K,T) = e^{-rT} \int_K^\infty (S_T - K) p(S_T) dS_T \tag{14.2}$$

To differentiate $C(S,K,T)$ with respect to $K$, we use Leibniz's Rule, which states:

**Theorem 14.1.1.** (Leibniz's Rule) *The derivative of a definite integral whose limits are functions of the differential variable is given by:*

$$\frac{\partial}{\partial z}\int_{a(z)}^{b(z)} f(x,z)dx = \frac{\partial b(z)}{\partial z}f(b(z),z) - \frac{\partial a(z)}{\partial z}f(a(z),z) + \int_{a(z)}^{b(z)} \frac{\partial}{\partial z}f(x,z)dx \tag{14.3}$$

Apply Leibniz's rule to evaluate

$$\frac{\partial}{\partial K}\int_K^\infty (S_T - K)p(S_T)dS_T$$

we get:

$$\frac{\partial}{\partial K} = -\int_K^\infty p(S_T)dS_T$$

Denoting $F(K)$ the cumulative density function of $S_T$,

$$\begin{aligned} e^{rT}\frac{\partial C(S,K,T)}{\partial K} &= = -\int_K^\infty p(S_T)dS_T \\ &= F(K) - 1 \end{aligned}$$

Differentiate again with respect to $K$ to get:

$$\frac{\partial^2 C}{\partial K^2}e^{rT} = p(K) \tag{14.4}$$

*Interpretation*

Consider a butterfly spread centered at $K$, and scaled to yield a maximum payoff of 1. Let $\phi(S_T)$ be the payoff function, illustrated in figure ?. For simplicity, we denote $C(S_T,K,T)$ as $C(K)$. Let $V$ be the value today:

$$\begin{aligned} V = & \frac{1}{\Delta K}[C(K+\Delta K) - 2C(K) + C(K-\Delta K)] \\ = & e^{-rT}\int_0^\infty \phi(S)p(S)dS \end{aligned} \tag{14.5}$$

In the interval $[K-\Delta K, K+\Delta K]$, approximate $p(S)$ by the constant $p(K)$ to get:

$$\begin{aligned} V = & e^{-rT}p(K)\int_0^\infty \phi(S)dS \\ = & e^{-rT}p(K)\Delta K \end{aligned} \tag{14.6}$$

Finally, use the definition of the derivative:

$$\lim_{\Delta K \to 0} \frac{1}{\Delta K^2} [C(K+\Delta K) - 2C(K) + C(K-\Delta K)] = \frac{\partial^2 C(K)}{\partial K^2} \tag{14.7}$$

to get:

$$p_T(K) = e^{rT} \frac{\partial^2 C(K)}{\partial K^2} \tag{14.8}$$

## 14.2 ANALYTICAL EXPRESSION FOR THE DENSITY OF $S_T$

Let $\sigma_T(K)$ be the implied volatility for maturity $T$, as a function of strike. If $\sigma_T(K)$ is twice differentiable with respect to $K$, then we can compute a closed form expression for the density of $S_T$. This result is due to Shimko (1993). To clearly distinguish the Black-Scholes model, which implies a constant volatility, from the model now under consideration, we denote:

$C^{BS}(K,\sigma)$ Price of a call, maturity $T$ and strike $K$, using the Black-Scholes model with constant volatility $\sigma$.

$C(K)$ Price of a call, maturity $T$ and strike $K$, with volatility function of strike: $\sigma(K)$.

*Preliminary calculations*

Let $n(x)$ be the normal density, and recall that, in the Black-Scholes standard notation,

$$\begin{aligned} n(d_2) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{d_2^2}{2}} \\ &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(d_1 - \sigma\sqrt{T})^2} \\ &= n(d_1) e^{d_1 \sigma\sqrt{T} - \frac{1}{2}\sigma^2 T} \\ &= n(d_1) \frac{S}{K} e^{rT}. \end{aligned} \tag{14.9}$$

We will also use the following intermediate result:

$$\frac{\partial d_2}{\partial K} = -\frac{1}{K\sigma\sqrt{T}} - \frac{d_1}{\sigma} \frac{\partial \sigma}{\partial K} \tag{14.10}$$

Next, differentiate the Black-Scholes call formula with respect to strike to get:

$$\frac{\partial C^{BS}(K,\sigma)}{\partial K} = S n(d_1) \frac{\partial d_1}{\partial K} - e^{-rT} K n(d_2) \frac{\partial d_2}{\partial K} - e^{-rT} N(d_2) \tag{14.11}$$

Observing that $\partial d_1 / \partial K = \partial d_2 / \partial K$, and using (14.9), one gets:

$$\frac{\partial C^{BS}(K,\sigma)}{\partial K} = -e^{-rT}N(d_2) \tag{14.12}$$

Next, let's differentiate $C(K)$ with respect to the strike. Remember that $C(K)$ is the price of a call of strike $K$, when volatility is itself a function of strike. Apply the chain rule to get:

$$\frac{\partial C(K)}{\partial K} = -e^{-rT}N(d_2) + \frac{\partial C^{BS}}{\partial \sigma}\frac{\partial \sigma}{\partial K} \tag{14.13}$$

where $\frac{\partial C^{BS}}{\partial \sigma}$ is the standard Black-Scholes vega:

$$\frac{\partial C^{BS}}{\partial \sigma} = Ke^{-rT}n(d_2)\sqrt{T}$$

Substituting in (14.13), one gets:

$$\frac{\partial C(K)}{\partial K} = -e^{-rT}N(d_2) + Ke^{-rT}n(d_2)\sqrt{T}\frac{\partial \sigma}{\partial K} \tag{14.14}$$

Differentiating again with respect to the strike:

$$\begin{aligned}\frac{\partial^2 C(K)}{\partial K^2} &= -e^{-rT}n(d_2)\frac{\partial d_2}{\partial K} + \\ &\quad e^{-rT}\sqrt{T}\left[n(d_2)\frac{\partial \sigma}{\partial K} + K\frac{\partial n(d_2)}{\partial d_2}\frac{\partial \sigma}{\partial K} + \right. \\ &\quad \left. Kn(d_2)\frac{\partial^2 \sigma}{\partial K^2}\right]\end{aligned} \tag{14.15}$$

where

$$\frac{\partial n(d_2)}{d_2} = -d_2 n(d_2)$$

After some algebra, and using (14.10), we get:

$$\begin{aligned}\frac{\partial^2 C(K)}{\partial K^2} = &-e^{-rT}n(d_2)\times \\ &\left\{\frac{1}{K\sigma\sqrt{T}} + \frac{\partial \sigma}{\partial K}\frac{2d_1}{\sigma} + \right. \\ &\left(\frac{\partial \sigma}{\partial K}\right)^2 \frac{\sqrt{T}Kd_1d_2}{\sigma} + \\ &\left.\frac{\partial^2 \sigma}{\partial K^2}K\sqrt{T}\right\}\end{aligned} \tag{14.16}$$

Combining (14.16) and the Breeden-Litzenberger formula (14.4) yields an expression for the density of $S_T$:

| Strike | Call | Put |
|---|---|---|
| 325.00 | 66.50 | 0.31 |
| 345.00 | 46.00 | 0.88 |
| 360.00 | 33.00 | 2.00 |
| 365.00 | 27.75 | 2.62 |
| 375.00 | 20.12 | 4.25 |
| 385.00 | 13.50 | 7.12 |
| 390.00 | 9.62 | 8.75 |
| 395.00 | 7.25 | 11.00 |
| 400.00 | 5.38 | 13.75 |
| 405.00 | 3.38 | 17.00 |
| 410.00 | 1.88 | 19.75 |
| 425.00 | 0.25 | 34.00 |

Table 14.1: Option prices, 60 days to expiry, for estimating a quadratic volatility model.

$$
\begin{aligned}
p(K) = & n(d_2) \times \\
& \left\{ \frac{1}{K\sigma\sqrt{T}} + \frac{\partial\sigma}{\partial K}\frac{2d_1}{\sigma} + \right. \\
& \left( \frac{\partial\sigma}{\partial K} \right)^2 \frac{\sqrt{T}Kd_1d_2}{\sigma} + \\
& \left. \frac{\partial^2\sigma}{\partial K^2} K\sqrt{T} \right\}
\end{aligned}
\tag{14.17}
$$

*Illustrations*

**Shimko's model** We illustrate the calculation with a quadratic form for $\sigma(K)$. The data is taken from Shimko (1993). This example will also illustrate the importance of correctly accounting for the smile when pricing exotic options - even the very simple ones.
The model is estimated from call and put prices for various strikes, listed in Table 14.1. The spot is 390.02 and the options expire in 60 days.

```
> K <- c(325, 345, 360, 365, 375, 385, 390, 395, 400, 405, 410,
    425)
> C <- c(66.5, 46, 33, 27.75, 20.125, 13.5, 9.625, 7.25, 5.375,
    3.375, 1.875, 0.25)
> P <- c(0.3125, 0.875, 2, 2.625, 4.25, 7.125, 8.75, 11, 13.75,
    17, 19.75, 34)
> T <- 60/360
> S0 <- 390.02
```

The first step is to compute the implied interest rate and dividend yield, using the call-put parity relationship:

$$F_T = (C_T - P_T)e^{rT} + K$$

with $F_T = S_0 e^{(r-d)T}$. Rearranging terms, we get:

$$S_0 e^{-dT} - K e^{-rT} = (C_T - P_T)$$

or,

$$C_T - P_T = \beta_0 + \beta_1 K$$

The coefficients $\beta_i$ are estimated by linear regression:

```
> mod <- lm(C - P ~ K)
> a0 <- as.numeric(mod$coefficients[1])
> a1 <- as.numeric(mod$coefficients[2])
> r <- -log(-a1)/T
> d <- log(S0/a0)/T
```

which yields $r = 6.8\%$ and $d = 4.98\%$. The next step is to compute the implied volatility for each strike:

```
> GBSPrice <- function(PutCall, S, K, T, r, b, sigma) {
     d1 <- (log(S/K) + (b + sigma^2/2) * T)/(sigma * sqrt(T))
     d2 <- d1 - sigma * sqrt(T)
     if (PutCall == "c")
         px <- S * exp((b - r) * T) * pnorm(d1) - K * exp(-r *
             T) * pnorm(d2)
     else px <- K * exp(-r * T) * pnorm(-d2) - S * exp((b - r) *
         T) * pnorm(-d1)
     px
 }
> GBSVega <- function(PutCall, S, K, T, r, b, sigma) {
     d1 <- (log(S/K) + (b + sigma^2/2) * T)/(sigma * sqrt(T))
     S * exp((b - r) * T) * dnorm(d1)
 }
> ImpliedVolNewton <- function(p, TypeFlag, S, X, Time, r, b, sigma = NULL,
     maxiter = 500, tol = 1e-05) {
     if (is.null(sigma))
         s <- sqrt(2 * abs(log(S/(X * exp((b * T))))))/T)
     else s <- sigma
     not_converged <- T
     i = 1
     vega <- GBSVega(TypeFlag, S, X, Time, r, b, s)
     while (not_converged & (i < maxiter)) {
         err <- (p - GBSPrice(TypeFlag, S, X, Time, r, b, s))
         s <- s + err/vega
         not_converged <- (abs(err/vega) > tol)
         i <- i + 1
     }
     s
 }
```

The function is used to compute the implied volatility for calls and puts:

```
> ivC <- sapply(seq_along(K), function(i) ImpliedVolNewton(C[i],
      "c", S0, K[i], T, r, b = r - d))
> ivP <- sapply(seq_along(K), function(i) ImpliedVolNewton(P[i],
      "p", S0, K[i], T, r, b = r - d))
```

We then fit a quadratic model to the implied volatility:

```
> lm.strike <- lm(c(ivC, ivP) ~ poly(rep(K, 2), 2, raw = TRUE))
> b1 <- lm.strike$coefficients[1]
> b2 <- lm.strike$coefficients[2]
> b3 <- lm.strike$coefficients[3]
> smileVol <- function(K) {
      b1 + b2 * K + b3 * K^2
  }
> bsVol <- function(K) median(c(ivC, ivP))
```

The actual and fitted values are displayed in Figure 14.1.

```
> plot(K, ivC, xlab = "Strike", ylab = "Volatility", col = "blue")
> lines(K, ivP, col = "green", type = "p")
> points(K, predict(lm.strike)[seq_along(K)], type = "l", col = "red",
      lwd = 3)
```

The density of $S_T$, according to a log-normal model is computed by:

```
> bs.pdf <- function(S, K, T, b, sigma) {
      d1 <- (log(S/K) + (b + sigma^2/2) * T)/(sigma * sqrt(T))
      d2 <- d1 - sigma * sqrt(T)
      sT <- sqrt(T)
      dnorm(d2)/(K * sigma * sT)
  }
```

while the density consistent with the quadratic smile is:

```
> shimko.pdf <- function(S, K, T, b, sigma) {
      d1 <- (log(S/K) + (b + sigma^2/2) * T)/(sigma * sqrt(T))
      d2 <- d1 - sigma * sqrt(T)
      sT <- sqrt(T)
      dsdK <- b2 + 2 * b3 * K
      d2sdK2 <- 2 * b3
      dnorm(d2) * (1/(K * sigma * sT) + dsdK * (2 * d1)/sigma +
          dsdK^2 * (K * d1 * d2 * sT)/sigma + d2sdK2 * K * sT)
  }
```

Figure 14.2 represents both densities. The log-normal density is computed with the average Black-Scholes volatility. The density implied from the quadratic volatility smile is computed by the Breeden-Litzenberger formula, and with Shimko's formula. The density implied by the smile curve clearly exhibit "fat tails".
The plot is constructed as follows:

```
> d2CdK2 <- function(vol, S, K, T, r, b) {
      dK <- K/10000
      c <- GBSPrice("c", S, K, T, r, b, vol(K))
      cPlus <- GBSPrice("c", S, K + dK, T, r, b, vol(K + dK))
```

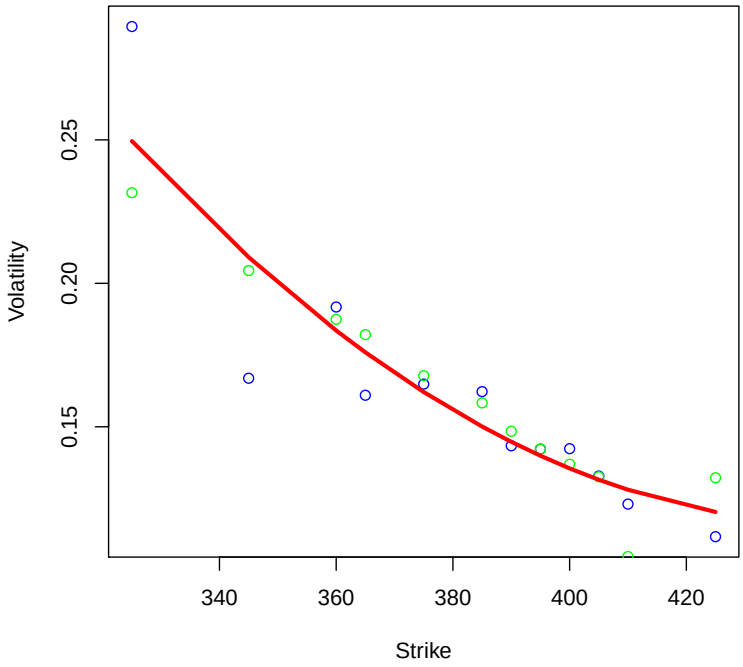



FIGURE 14.1: Quadratic model fitted to call (blue) and put(green) implied volatility

```
    cMinus <- GBSPrice("c", S, K - dK, T, r, b, vol(K - dK))
    (cPlus - 2 * c + cMinus)/(dK^2)
}
> KRange <- seq(300, 500, 1)
> nb <- length(KRange)
> p <- matrix(nrow = nb, ncol = 3)
> p[, 1] <- sapply(KRange, function(K) {
    bs.pdf(S0, K, T, b = r - d, bsVol(K))
})
> p[, 2] <- sapply(KRange, function(K) {
    d2CdK2(smileVol, S0, K, T, r, b = r - d) * exp(r * T)
})
> p[, 3] <- sapply(KRange, function(K) {
    shimko.pdf(S0, K, T, b = r - d, smileVol(K))
})
> cl <- c("green", "red", "blue")
> plot(KRange, p[, 1], main = "Implied Density from Smile", type = "l",
    xlab = "Strike", ylab = "p(K)", col = cl[1], ylim = c(min(p),
        max(p)))
```

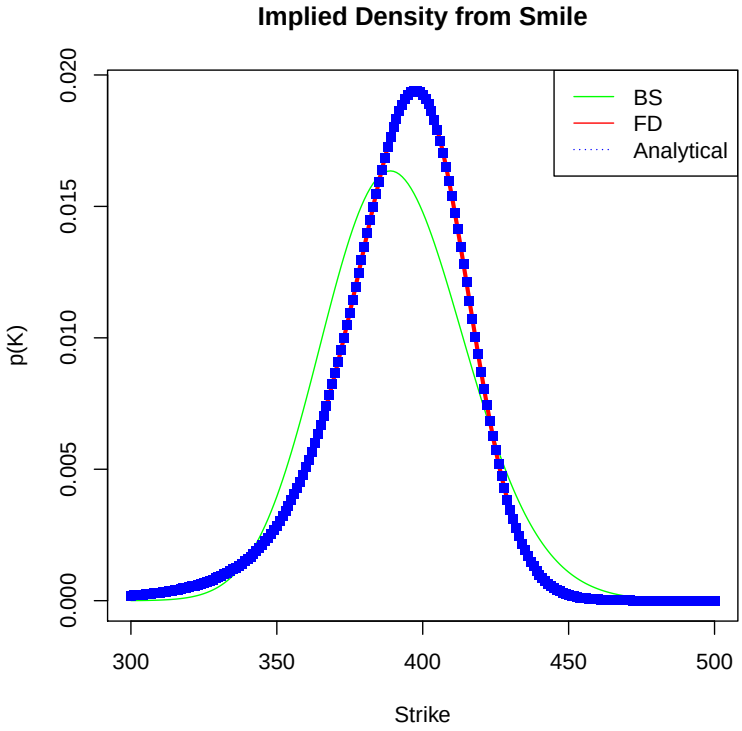



FIGURE 14.2: Density of $S_T$, with constant volatility and quadratic model for implied volatility.

```
> lines(KRange, y = p[, 2], type = "l", lwd = 3, col = cl[2])
> lines(KRange, y = p[, 3], type = "p", pch = 15, lwd = 1, col = cl[3])
> legend("topright", c("BS", "FD", "Analytical"), lty = c(1, 1,
      15), col = cl)
```

To illustrate the importance of correctly accounting for the volatility smile, we now consider a digital option maturing at the same time as our European options. We want to price this option in a way that is consistent with the observed volatility smile, modeled by a quadratic function.
The payoff function is:

```
> digital.payoff <- function(S) ifelse(S > Kd, 1, 0)
```

A naive approach would be to look up the Black-Scholes volatility corresponding to the strike, and price the digital option accordingly. The price of a digital cash-or-nothing call would then be given by:

$$C = e^{-rT}\Phi(d_2) \tag{14.18}$$

with:

$$\begin{aligned}\Phi(x) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}x^2} \\ d_2 &= \frac{\ln\frac{S}{K}+(r-d-\frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\end{aligned}$$

The following function implement this calculation:

```
> BS.digital.call <- function(K) {
     sigma <- smileVol(K)
     d2 <- (log(S0/K) + (r - d - (sigma^2/2)) * T)/(sigma * sqrt(T))
     pnorm(d2) * exp(-r * T)
 }
```

However, since we know the risk-neutral density of $S_T$, we can directly compute the expected discounted value of the digital payoff:

```
> digital.shimko <- function(K) {
     digital.payoff(K) * shimko.pdf(S0, K, T, b = r - d, smileVol(S0))
 }
> shimko.digital.call <- function(K) {
     exp(-r * T) * integrate(Vectorize(digital.shimko), lower = K,
         upper = 700)$value
 }
```

For comparison, an at-the-money call is evaluated according to both methods:

```
> Kd <- S0
> price.BS <- BS.digital.call(S0)
> price.shimko <- shimko.digital.call(S0)
```

which yields a price of 50.3% with the Black-Scholes model and 0.6% with Shimko's density. The Black-Sholes model with an interpolated volatility mis-prices the digital call by more than 10%. Figure 14.3 compares the prices obtained by both methods, for a range of strikes.

```
> KdRange <- seq(350, 450, 5)
> BS.Price <- sapply(KdRange, function(K) BS.digital.call(K))
> Shimko.Price <- sapply(KdRange, function(K) shimko.digital.call(K))
```

*WTI Nymex Volatility*

In the spring of 2009, in the wake of a severe correction in the crude oil market, the volatility smile for the December 2009 NYMEX options on the WTI Futures contract has a pronounced asymmetry. We will use the Breeden-Litzenberger formula to infer the density the Futures price at expiry. The volatility smile as a function of Quick Delta is presented in Figure 14.4.

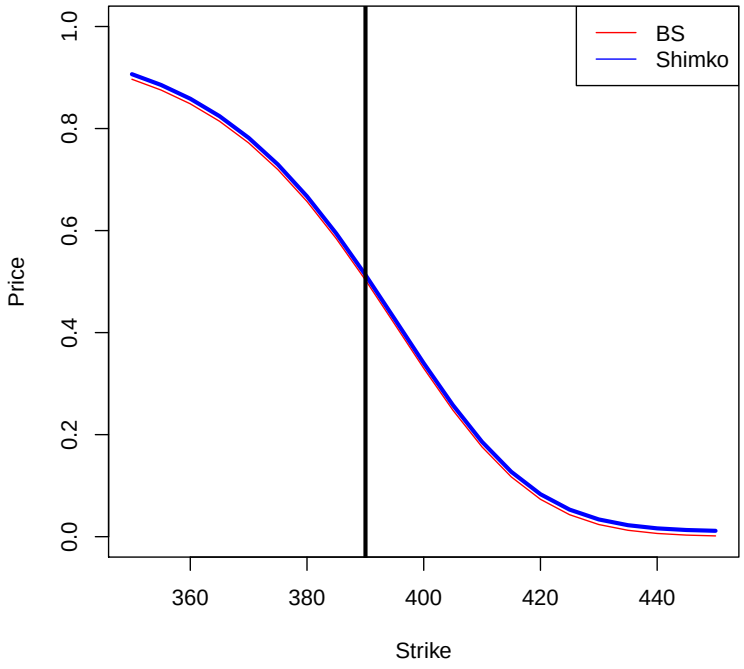



FIGURE 14.3: Comparison of prices of a digital option, using a log-normal density for $S_T$, and using the density implied by the volatility smile fitted to a quadratic function.

The dataset `CL.ImpliedVol` may be found in packageempfin. The variables of interest for the December 2009 contract are extracted as follows:

```
> indx <- CL.ImpliedVol$table[, "DtContract"] == as.Date("2009-12-01")
> iv <- CL.ImpliedVol$table[indx, "IV"]
> QD <- CL.ImpliedVol$table[indx, "QuickDelta"]
> Fwd <- CL.ImpliedVol$table[indx, "Fwd"][1]
> T <- CL.ImpliedVol$table[indx, "T"][1]
> ATMVol <- CL.ImpliedVol$table[indx, "ATMVol"][1]
> r <- CL.ImpliedVol$table[indx, "IR"][1]
```

The calculation of implied volatility by (14.4) requires a smooth smile curve. The first step is thus to fit a model to the observed implied volatility. We consider 3 linear model, with strike, log-moneyness or quick delta as independent variable:

```
> LogMoneyness <- qnorm(QD) * ATMVol * sqrt(T)
> Strike <- Fwd/exp(LogMoneyness)
```

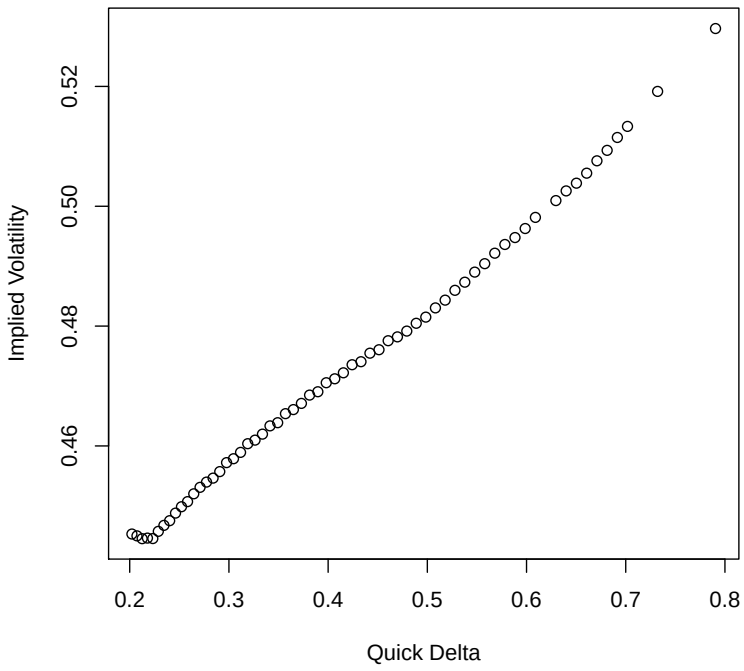



FIGURE 14.4: Implied volatility of WTI NYMEX options on the December 2009 Futures, observed on April 21, 2009

Implied volatility as a function of strike:

```
> lm.Strike <- lm(iv ~ Strike)
> smileVol.Strike <- function(K) {
      predict(lm.Strike, data.frame(Strike = K))
  }
```

Implied volatility as a function of log-moneyness:

```
> lm.LogMoneyness <- lm(iv ~ LogMoneyness)
> smileVol.LogM <- function(K) {
      LogM <- log(Fwd/K)
      predict(lm.LogMoneyness, data.frame(LogMoneyness = LogM))
  }
```

Implied volatility as a function of Quick-Delta:

```
> lm.QD <- lm(iv ~ QD)
> smileVol.QD <- function(K) {
      QD <- pnorm(log(Fwd/K)/(ATMVol * sqrt(T)))
```

```
    predict(lm.QD, data.frame(QD = QD))
}
```

The implied density corresponding to each model is computed next, using the Breeden-Litzenberger formula.
The second derivative $\frac{\partial^2 C}{\partial K^2}$ is computed by finite difference:

```
> d2CdK2 <- function(vol, S, K, T, r, b = 0) {
    dK <- 1e-04
    c <- GBSOption("c", S, K, T, r, b, vol(K))@price
    cPlus <- GBSOption("c", S, K + dK, T, r, b, vol(K + dK))@price
    cMinus <- GBSOption("c", S, K - dK, T, r, b, vol(K - dK))@price
    (cPlus - 2 * c + cMinus)/(dK^2)
}


> KRange <- seq(10, 150, 1)
> nb <- length(KRange)
> p <- matrix(nrow = nb, ncol = 4)
> b <- 0
> bsVol <- function(K) {
    ATMVol
}
> p[, 1] <- sapply(KRange, function(K) d2CdK2(bsVol, Fwd, K, T,
    r, b) * exp(r * T))
> p[, 2] <- sapply(KRange, function(K) d2CdK2(smileVol.Strike,
    Fwd, K, T, r, b) * exp(r * T))
> p[, 3] <- sapply(KRange, function(K) d2CdK2(smileVol.LogM, Fwd,
    K, T, r, b) * exp(r * T))
> p[, 4] <- sapply(KRange, function(K) d2CdK2(smileVol.QD, Fwd,
    K, T, r, b) * exp(r * T))
```

The implied densities are represented in Figure 14.5.
At this stage, we have computed the density at a sample of strikes. We next fit a density function to this data set. Using density from the quick delta model as an example, a preliminary step is of course to insure that the discreet probabilities sum to 1.

```
> pr <- p[, 4]
> pr <- pr/sum(pr)
```

We fit the data to a Johnson density by matching the first 4 moments. The first moment is set to 0 since this is the risk-neutral density of a Futures, which should have no drift.

```
> KRange <- KRange * Fwd/sum(pr * KRange)
> Rd <- log(KRange/Fwd)
> m1 <- sum(pr * Rd)
> m2 <- sum(pr * (Rd - m1)^2)
> m3 <- sum(pr * (Rd - m1)^3)
> m4 <- sum(pr * (Rd - m1)^4)
> J.fit <- JohnsonFit(c(m1, m2, m3, m4), moment = "use")
```

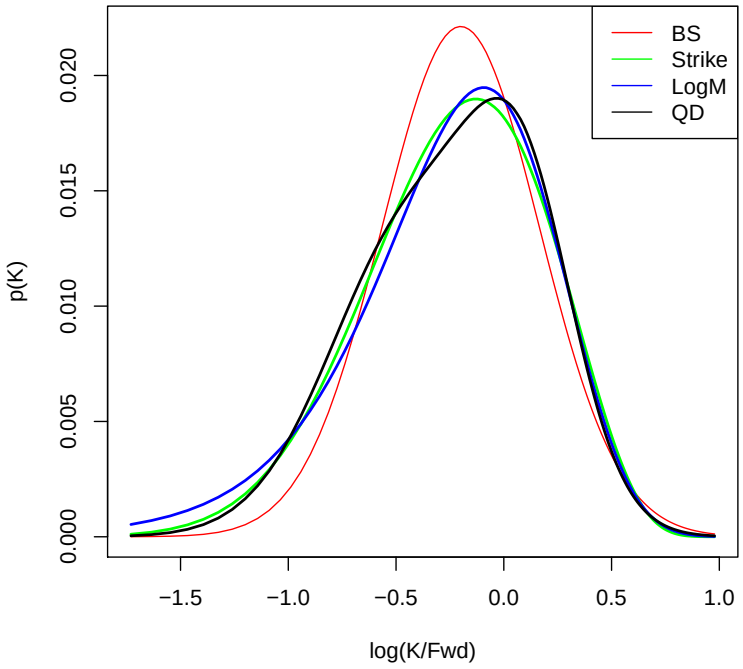



FIGURE 14.5: Density of WTI Dec. 2009 Futures at expiry, implied by option smile

The empirical and fitted density functions are represented in Figure 14.6. The fitted distribution could be used to price European derivatives that are function of the Futures price at expiry.

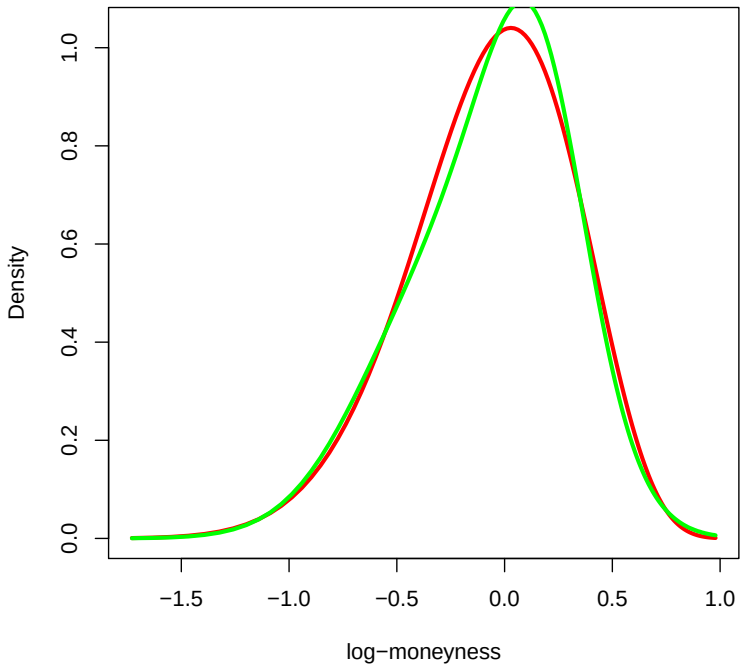



FIGURE 14.6: Empirical and fitted implied density of WTI Futures price

# Chapter 15

# Implied Trees

```
> library(fOptions)
> library(fExoticOptions)
> library(fAsianOptions)
> library(fInstrument)
> library(empfin)
```

The Breeden-Litzenberger formula provides the density of the underlying asset at expiry as a function of vanilla prices. In many instances, for pricing American options for example, it is important to know the density at any time $t$ between the current time and expiry. In this chapter, we construct the function $\sigma(S_t, t)$, i.e. the instantaneous volatility as a function of the underlying asset and time.

## 15.1 The Derman-Kani Implied Tree

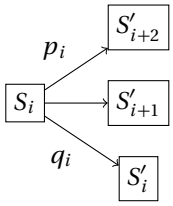
In a standard trinomial tree, the transition probabilities are independent of the state. By contrast, the Derman-Kani implied tree involves transition probabilities that are state-dependent. With this extension, one obtain a model that matches the observed prices of vanilla options at various maturities and strikes.

### *Principle*

We start with a trinomial tree constructed with a constant volatility. This determines the geometry of the tree, according to the formulae derived in Section 7.7. We will now modify the transition probabilities in order to match observed option prices.
Let's focus on one node and its three children nodes. The notation is illustrated in Figure 15.1.

-1

FIGURE 15.1: One step of a trinomial tree




-1

States $S_i$ are associated with maturity $T$, and states $S'_i$ with maturity $T+\Delta t$.
We will also use the following notation:

$\lambda_i$ state price for node $i$

$C(S'_i)$ call price, strike $S'_i$, maturity $T+\Delta t$

The call can be priced in the trinomial tree, giving:

$$\begin{aligned} C(S'_i) &= \lambda_i e^{-r\Delta t} p_i (S'_{i+2} - S'_{i+1}) \\ &\quad + \sum_{j=i+1}^{2n} \lambda_j e^{-r\Delta t} \left[ p_j (S'_{j+2} - S'_{i+1}) + (1 - p_j - q_j)(S'_{j+1} - S'_{i+1}) + q_j (S'_j - S'_{i+1}) \right] \end{aligned}$$

The price process in the tree is a martingale, and thus:

$$\begin{aligned} S'_j e^{r\Delta t} &= F_j \\ &= p_j S'_{j+2} + (1 - p_j - q_j) S'_{j+1} + q_j S'_j - S'_{i+1} \end{aligned}$$

Using this identiy, the call price becomes:

$$\begin{aligned} C(S'_i) &= \lambda_i e^{-r\Delta t} p_i (S'_{i+2} - S'_{i+1}) \\ &\quad + \sum_{j=i+1}^{2n} \lambda_j e^{-r\Delta t} \left[ F_j - S'_{i+1} \right] \end{aligned}$$

In the equation above, all quantities except $p_i$ are known, yielding:

$$p_i = \frac{e^{r\Delta t} C(S'_{i+1}) - \sum_{j=i+1}^{2n} \lambda_j e^{-r\Delta t} \left[ F_j - S'_{i+1} \right]}{\lambda_i (S'_{i+2} - S'_{i+1})} \tag{15.1}$$

Using again the martingale property, one can compute the probability $q_i$:

$$q_i = \frac{F_i - p_i (S'_{i+2} - S'_{i+1}) - S'_{i+1}}{S'_i - S'_{i+1}} \tag{15.2}$$

The corresponding local volatility at node $S'_i$ is finally obtained by:

$$\begin{aligned} &\sigma(S'_i, T) F_i^2 \Delta t = \\ &\quad p_i (S'_{i+2} - F_i)^2 + (1 - p_i - q_i)(S'_{i+1} - F_i)^2 + q_i (S'_i - F_i)^2 \end{aligned} \tag{15.3}$$

A similar calculation can be performed with put prices.

*Illustration*

For the sake of this example, we will use a simple expression for the Black-Scholes volatility, function of time to expiry ($T$) and spot price ($S$):

$$\sigma(S,T) = \sigma_0(1 + a(S - 100)) + bT \tag{15.4}$$

```
> sigma <- 0.3
> a <- -0.2/100
> b <- 0
> bsvol <- function(S, T) {
      sigma * (1 + a * (S - 100)) + b * T
  }
```

We now proceed step-by-step in the calculation process. The first step is to construct a classical trinomial tree, with constant probabilities. This determines the geometry of the tree.
In this example, we construct a 3-periods tree, anual volatility is 30%. For simplicity, interest rate and dividend yield is set to 0.

```
> r <- 0
> b <- 0
> n <- 3
> dt <- 0.01
> nr <- 2 * (n + 1) - 1
> nc <- (n + 1)
> T <- seq(0, nc) * dt
> u <- exp(sigma * sqrt(2 * dt))
> d <- 1/u
> pu <- ((exp(r * dt/2) - exp(-sigma * sqrt(dt/2)))/(exp(sigma *
      sqrt(dt/2)) - exp(-sigma * sqrt(dt/2))))^2
> pd <- (1 - sqrt(pu))^2
> pm <- 1 - pu - pd
> S <- matrix(data = NA, nrow = nr, ncol = nc)
> for (j in seq(1, nc)) {
      S[1:(1 + 2 * (j - 1)), j] <- 100 * u^(j - 1) * d^seq(0, 2 *
          (j - 1))
  }
```

The resulting tree is pictured in Figure 15.2. This is a standard trinomial tree. Branching probabilities are identical at each node, volatility at each node is also constant, 30% annual rate.
The next step is to price calls and puts expiring at each time step, and struck at each mode value. The volatility used to price each option is time and strike dependent, per equation (15.4).
The volatility grid is computed by evaluating the volatility function for each node:

```
> Vol <- matrix(data = NA, nrow = nr, ncol = nc)
> for (i in seq(1, nr)) {
      for (j in seq(1, nc)) if (!is.na(S[i, j]))
          Vol[i, j] <- bsvol(S[i, j], T[j])
```

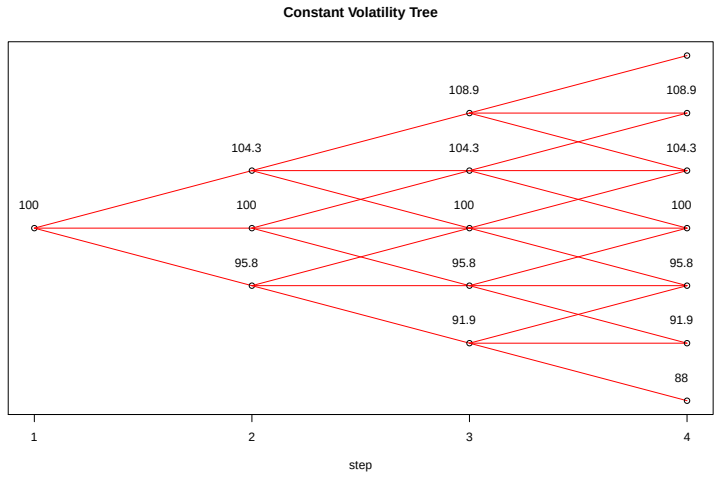



FIGURE 15.2: Constant Volatility Trinomial Tree

```
    }
```

and this volatility is used to compute the call and put prices expiring at each time step and strike level:

```
> Call <- matrix(data = NA, nrow = nr, ncol = nc)
> Put <- matrix(data = NA, nrow = nr, ncol = nc)
> for (i in seq(1, nr)) {
     for (j in seq(2, nc)) if (!is.na(S[i, j])) {
         Call[i, j] <- CRRTrinomial("ce", S[1, 1], S[i, j], T[j],
             r, b, Vol[i, j], j - 1)$price
         Put[i, j] <- CRRTrinomial("pe", S[1, 1], S[i, j], T[j],
             r, b, Vol[i, j], j - 1)$price
     }
 }
```

Next, we use equations (15.1) and (15.2) to compute the transition probabilities at each node. The probability $p_i$ is computed by:

```
> p <- function(i, j) {
     SP1 <- S[i + 1, j + 1]
     SP2 <- S[i, j + 1]
     tmp = 0
     if (i > 1) {
         l <- Lambda[1:(i - 1), j]
         F <- S[1:(i - 1), j] * exp(r * dt)
         tmp = t(l) %*% (F - SP1)
     }
     (exp(r * dt) * Call[i + 1, j + 1] - tmp)/(Lambda[i, j] *
         (SP2 - SP1))
 }
```

and the probability $q_i$ is determined by:

```
> q <- function(i, j) {
     SP2 <- S[i, j + 1]
     SP1 <- S[i + 1, j + 1]
     SP <- S[i + 2, j + 1]
     F <- S[i, j] * exp(r * dt)
     (F - p(i, j) * (SP2 - SP1) - SP1)/(SP - SP1)
 }
```

With these two functions, we can proceed recursively, computing the state prices and the transition probabilities one time step at a time. Note that functions $p$ and $q$ above use as input the state prices up to time step $i-1$ in order to compute the transition probabilities at time step $i$. The transition probabilites for the root node are computed immediately:

```
> Lambda <- matrix(data = NA, nrow = 7, ncol = 4)
> Lambda[1, 1] <- 1
> Pu <- p(1, 1)
> Pd <- q(1, 1)
> Pm <- 1 - Pu - Pd
```

and this provides the data for computing the state prices $\lambda_{i,2}, i = 1,\ldots,3$ for time step $\Delta t$.

```
> Lambda[1, 2] <- Pu * exp(-r * dt)
> Lambda[2, 2] <- Pm * exp(-r * dt)
> Lambda[3, 2] <- Pd * exp(-r * dt)
```

The state prices for the other time steps are computed similarly.

```
> Lambda[1, 3] <- p(1, 2) * Lambda[1, 2]
> Lambda[2, 3] <- (1 - p(1, 2) - q(1, 2)) * Lambda[1, 2] + p(2,
     2) * Lambda[2, 2]
> Lambda[3, 3] <- q(1, 2) * Lambda[1, 2] + (1 - p(2, 2) - q(2,
     2)) * Lambda[2, 2] + p(3, 2) * Lambda[3, 2]
> Lambda[4, 3] <- (1 - p(3, 2) - q(3, 2)) * Lambda[3, 2] + q(2,
     2) * Lambda[2, 2]
> Lambda[5, 3] <- q(3, 2) * Lambda[3, 2]
> Lambda[1, 4] <- p(1, 3) * Lambda[1, 3]
> Lambda[2, 4] <- (1 - p(1, 3) - q(1, 3)) * Lambda[1, 3] + p(2,
     3) * Lambda[2, 3]
> Lambda[3, 4] <- q(1, 3) * Lambda[1, 3] + (1 - p(2, 3) - q(2,
     3)) * Lambda[2, 3] + p(3, 3) * Lambda[3, 3]
> Lambda[4, 4] <- q(2, 3) * Lambda[2, 3] + (1 - p(3, 3) - q(3,
     3)) * Lambda[3, 3] + p(4, 3) * Lambda[4, 3]
> Lambda[5, 4] <- q(3, 3) * Lambda[3, 3] + (1 - p(4, 3) - q(4,
     3)) * Lambda[4, 3] + p(5, 3) * Lambda[5, 3]
> Lambda[6, 4] <- (1 - p(5, 3) - q(5, 3)) * Lambda[5, 3] + q(4,
     3) * Lambda[4, 3]
> Lambda[7, 4] <- q(5, 3) * Lambda[5, 3]
```

Since interest rate is 0, the state prices at each time step should sum up to 1. This is verified by:

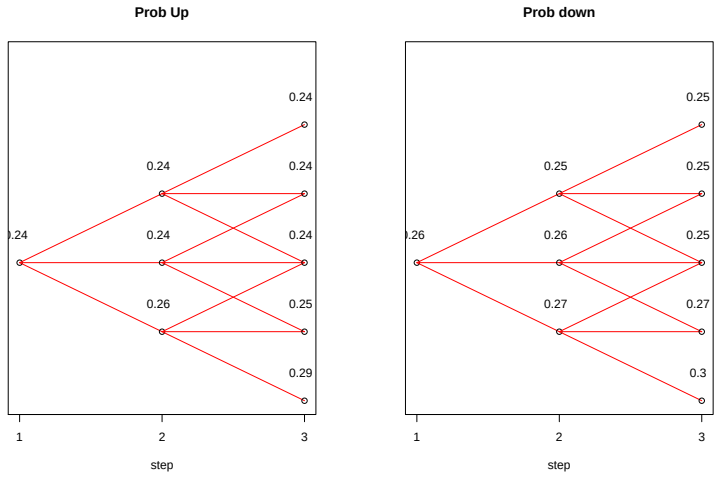



FIGURE 15.3: Transition probabilities in Derman-Kani implied tree

```
> z <- apply(Lambda, 2, function(x) {
      sum(x[!is.na(x)])
  })
> print(z)
[1] 1 1 1 1
```

Having determined the state prices, we record the transition probabilites in grids, in order to facilitate the display. The up and down probabilities associated with each node are displayed in Figure 15.3.

```
> Pup <- matrix(data = NA, nrow = nr, ncol = nc)
> Pdn <- matrix(data = NA, nrow = nr, ncol = nc)
> Pmd <- matrix(data = NA, nrow = nr, ncol = nc)
> for (i in seq(1, nr)) {
      for (j in seq(1, nc - 1)) if (!is.na(S[i, j])) {
          Pup[i, j] <- p(i, j)
          Pdn[i, j] <- q(i, j)
          Pmd[i, j] <- 1 - Pup[i, j] - Pdn[i, j]
      }
  }
```

Finally, the local volatility at each node can be computed from the transition probabilities:

```
> lvol <- function(i, j) {
      SP2 <- S[i, j + 1]
      SP1 <- S[i + 1, j + 1]
      SP <- S[i + 2, j + 1]
      F <- S[i, j] * exp(r * dt)
      sqrt((Pup[i, j] * (SP2 - F)^2 + Pdn[i, j] * (SP - F)^2 +
```

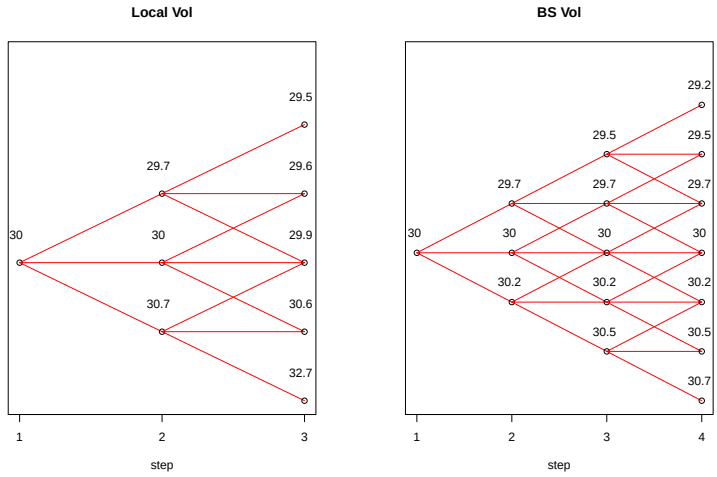



FIGURE 15.4: BlackScholes and local volatilities

```
        Pmd[i, j] * (SP1 - F)^2)/(F^2 * dt))
 }
> LVol <- matrix(data = NA, nrow = nr, ncol = nc)
> for (i in seq(1, nr)) {
     for (j in seq(1, nc - 1)) if (!is.na(S[i, j])) {
         LVol[i, j] <- lvol(i, j)
     }
 }
```

Figure 15.4 shows the local volatility associated with each node, and, for comparison, the Black-Scholes volatility, which is the average volatility from time 0 to expiry associated with a particular strike and expiry date.

# BIBLIOGRAPHY


Breeden, D. T. & Litzenberger, R. H. (1978). Prices of state-contingent claims implicit in option prices. *Journal of Business, Oct.*, 621–651.

Cox, J., Ross, S., & Rubinstein, M. (1979). Option pricing: a simplified approach. *Journal of Financial Econometrics, 7*, 229–263.

Eddelbuettel, D. (2009). Rquantlib.

Fowler, M. (1996). *Analysis Patterns: Reusable Object Models*. Addison-Wesley.

Genolini, C. (2008). A (not so) short introduction to s4.

Haug, E. G. (2006). *The Complete Guide to Option Pricing Formulas*. McGraw Hill.

Hull, J. (1997). *Options, Futures and Other Derivatives*. Prentice Hall.

Jackel, P. (2006). By implication. *Wilmott Magazine*.

Joshi, M. S. (2007). The convergence of binomial trees for pricing the american put.

Karoui, N., Jeanblanc-Picquè, M., & Shreve, S. (1998). Robustness of the black and scholes formula. *Mathematical Finance, 8*(2), 93–126.

Leisen, D. & Reimer, M. (1996). Binomial models for option valuation - examining and improving convergence. *Applied Mathematical Finance, 3*, 319–346.

Leland, H. (1985). Option pricing and replication with transactions costs. *The Journal of Finance, 40*(5), 1283–1301.

Nelson, C. & Siegel, A. (1987). Parsimonious modeling of yield curves. *Journal of Business, 60*, 473–490.

R. Jarrow, A. R. (1993). *Option pricing*. Richard D. Irwin.

Shimko (1993). Bounds on probability. *Risk, 6*, 33–37.

Shkolnikov, Y. (2009). Generalized vanna-volga method and its applications. *NumeriX Research Paper*.

Svensson, L. E. (1994). Estimating and interpreting forward interest rates: Sweden 1992-1994. *IMF Working Paper, 94/114*.

# INDEX


base.env, 17
Bond, 32

DataProvider, 13

environment, 9, 13

fInstrument.r, 11
fInstrumentFactory, 10, 11
fInstrumentfactory, 11

GBSOption, 12
getValue, 12

keywords
  all, 143
  apply, 8, 143
  as, 7, 8, 143, 146
  c, 6–8
  character, 7
  close, 6, 8
  colnames, 6
  csv, 5, 6, 8, 143
  data, 6, 8, 143, 146
  df, 143, 146
  dim, 8
  dt, 8, 146
  env, 12
  file, 143, 146
  for, 8, 146
  format, 7, 8
  frame, 6, 8, 146
  function, 5, 7, 8, 11, 12, 143
  get, 7, 8
  grep, 8
  if, 6, 12
  is, 143
  length, 8, 143
  load, 146
  match, 143
  na, 143
  names, 8
  ncol, 8
  numeric, 8, 143
  parse, 143
  paste, 5, 7, 12, 143, 146
  path, 143
  print, 12, 143, 146
  rbind, 146
  read, 6, 8, 143
  return, 12
  row, 8
  sapply, 8
  save, 143
  seq, 8
  strsplit, 5, 8
  subset, 146
  switch, 11, 12
  symbol, 143
  t, 12
  trace, 12
  ts, 8
  unique, 146
  url, 5, 7, 8
  write, 143

makeTable, 17

pathSimulator, 15

R classes
DataProvider, 3, 9, 11, 13, 14, 17–19
environment, 19
fInstrument, 3, 9–12, 18, 79
finstrument, 11
timeSeries, 15, 18
timeseries, 7
R data
CL-Historical, 7
CL.ImpliedVol, 159
ECBYieldCurve, 34
EquitySeries, 8
FedYieldCurve, 34
LiborRates, 8, 43
R functions
compute.iv, 145
R packages
DynamicSimulation, 3, 16
empfin, 3, 5, 7, 8, 32, 146
fInstrument, 3, 11
YieldCurve, 34, 39

Vanilla.r, 12

ZCRate, 37

# About the Author

**Patrik Hénaff** is Associate Professor at Université Paris-I (Panthéon-Sorbonne), where he teaches market finance to MBA students. He also taught at UBO (Université de Bretagne Occidentale) Telecom Bretagne, both situated in Brest, France.
Prior to this, he spent many years in investment banking, in France and in the US, working in various quantitative research roles.

Patrick Hénaff is a graduate of the Ecole des Hautes Etudes Commerciales (HEC) in Paris, and holds a PhD from the University of Texas.