A

Programming in VBA

The purpose of this appendix is to provide an introduction to the features of Excel VBA that are used in the book. To learn more about VBA in a finance setting, Jackson and Staunton [40] is a good source.

A.1 VBA Editor and Modules

Subroutines and functions are created in the VBA editor, which is reached through Tools/ Macros/Visual Basic Editor. When the editor opens, click Insert/Module to open an editing screen in which subroutines and functions can be typed. If you have opened a new workbook and inserted a module, you should see on the left a small pane with the heading Project-VBA Project that lists the elements of the workbook, including Module 1, which is the default name for the collection of things you might type in the editing screen (if the pane is not present, click View/Project Window). You should also see on the left a pane with the heading Properties-Module 1 (if it is not there, click View/Properties). You can rename Module 1 to something more useful by highlighting Module 1 in the Properties Window and typing the new name. You can add another module by clicking Insert/Module again. If you save the Excel workbook, all of the modules (and hence all of the subroutines and functions created in them) are saved with the workbook.

If you open the workbook distributed with this book, you will see in the Project Window modules named Chapt2, Chapt3, ..., Chapter 13. Each of these modules contains the VBA code in the corresponding book chapter. To view the code in a particular module, right-click on its name in the Project Window and select View Code. You will see a collection of programs separated by gray lines (which are added by the VBA editor to make things more readable). Each subroutine starts with `Sub` and ends with `End Sub` and each function starts with `Function` and ends with `End Function`. You will also see "Option Explicit" at the top of each module—this will be discussed below.

The organization of subroutines and functions into modules is not important (except for globally defined variables, which are not used in this book). All of the subroutines and all of the functions in all of the modules in any open workbook are available to use in any open workbook. However, you may find it convenient to organize your work into separate modules, for example Homework1, Homework2, etc.

If you open multiple workbooks, and open the VBA editor with one of them, then the Project Window will list each workbook and the modules associated with each. When the workbooks are saved, each set of modules will be saved with the associated workbook.

If VBA catches an error (normally a syntax error or an undeclared variable if "Option Explicit" has been declared) when executing a subroutine or function, a message box will pop up to inform the user. If the "Debug" option is chosen in this box, the offending VBA code will be highlighted in the VBA editor. After correcting the error, you need to click Run/Reset in the editor (or the square button in the editor's toolbar).

VBA ignores everything written on a line following an apostrophe, so comments can be placed on any line by preceding them with an apostrophe. Including comments in your subroutines and functions is very important to make them understandable.

The underscore character indicates that a line is to be continued. For example,

```
y = x + 5
```

is the same as

```
y = x _
+ 5
```

This is useful for breaking long lines.

A.2 Subroutines and Functions

A subroutine is also called a macro. It is a way of automating tasks, including mathematical calculations, cell formatting, and outputting results to cells. The subroutines in this book simulate a random process and output the results to the active worksheet. The other programs in the book are user-defined functions.

To execute a macro, click Tools/Macros. Clicking the name of a macro and then clicking Run will execute it. A macro or function created in one workbook can be used in another. To execute a macro created in another workbook, simply open both workbooks at the same time, click Tools/Macros and choose the option "All Open Workbooks" for the macros to be displayed.[1]

[1] Since you will be running macros and using user-defined functions frequently, it is useful to add buttons to the toolbar to execute the keystrokes of clicking

To create a macro in the VBA editor, type

```
Sub WhateverNameYouWant()
...
list of commands
...
End Sub
```

You will notice that the editor automatically adds the parentheses `()` at the end of the subroutine name and adds the `End Sub` statement when you type `Sub WhateverNameYouWant`.

A user-defined function is executed just like any other Excel function—in a cell of the spreadsheet, type `=FunctionName(arguments)`. The arguments supplied to the functions can be numbers or can be cell references, just as with any other Excel function. To see the user-defined functions that have been created, click Insert/Function and select the category User Defined. You may see a lot of functions created by Excel add-ins in addition to the functions that are in the modules. You can also execute a function by double-clicking on its name here.

To create a function in the VBA editor, type

```
Function AnotherName(argument1, argument2, ..., lastargument)
...
list of commands
...
AnotherName = WhateverTheAnswerMightBe
End Function
```

A.3 Message Box and Input Box

One way for a subroutine or function to deliver information is through the `MsgBox` function. In Module 1, type

```
Sub WhateverNameYouWant()
MsgBox("Whatever you want to type.")
End Sub
```

When you execute this macro, a message box will pop up, displaying the message. To close the message box, click OK. The message box function is useful primarily for displaying error messages. However, the message box can also return the results of mathematical operations, as the next example shows.

Tools/Macros and Insert/Function, if the buttons are not already there. To add the macro button, click Tools/Customize/Commands/Tools, scroll to Macros, and drag the "Macros ..." button to the toolbar. To add the function button, scroll to Insert and drag the "Insert Function" button to the toolbar.

One way for a subroutine or function to obtain information from the user is via the `InputBox` function. In Module 1, type

```
Sub AnotherSub()
x = InputBox("What is your favorite number?")
MsgBox("You said your favorite number was  " & x)
End Sub
```

When you execute this macro, a box will pop up displaying the text "What is your favorite number?" and providing a facility for inputting a number. When you hit Enter or click OK, the input box will disappear and the message box will appear, displaying the message and the number you chose.

A.4 Writing to and Reading from Cells

You can write a number, text, or formula to any cell in any worksheet of any open workbook. For example, executing the following macro

```
Sub WritingTest()
Workbooks("Book1.xls").Sheets("Sheet1").Range("B3").Value = 7
End Sub
```

will write the number 7 to cell B3 of Sheet1 of Book1.xls. The statement can be shortened to

```
Sheets("Sheet1").Range("B3").Value = 7
```

if you want to write to the active workbook, and it can be shortened to

```
Range("B3").Value = 7
```

if you want to write to the active sheet in the active workbook. To write text to the cell, enclose it in parentheses; for example, we could replace `Value = 7` with `Value = "some text"`. It is also possible to write a formula to a cell by replacing `Value = 7` with, for example, `Formula = "=A6"`. Running any macro of this sort will over-write anything that may already be in cell B3.

In the macros in this book, rather than writing to a particular cell, we write to the active cell of the active sheet of the active workbook (i.e., the cell in which the cursor is) and to cells surrounding the active cell. This is done as follows:

```
Sub WritingTest()
ActiveCell.Value = 7
ActiveCell.Offset(1,2) = 8
End Sub
```

This macro writes the number 7 to the active cell and the number 8 to the cell that is one row below and two columns to the right of the active cell.

A subroutine or function can also read directly from a cell in a workbook, though we do not use that feature in this book. The syntax is the same as for writing to a cell; for example, `x = ActiveCell.Value` assigns the value in the active cell to the variable `x`.

The formatting of cells (and ranges of cells) can also be changed in Excel macros. Moreover, the active cell/sheet/workbook can also be selected within a macro, and charts can be generated within macros, etc. We use VBA mainly as a computational engine in this book rather than as a means to create and modify worksheets, so we do not use many of the features of Excel VBA.

A.5 Variables and Assignments

Variable names must begin with a letter, be less than 256 characters long, and cannot include various special characters (in particular, they cannot contain blank spaces, hyphens or periods). Variable names are not case sensitive: `a` is the same variable as `A` (in fact, you may find the VBA editor changing the capitalization of names to maintain consistency across a project). It is of course a good idea to use names that mean something, so your programs are easier to read later. You cannot use any name already reserved by VBA or Excel; for example, attempting to create a variable with the name `Sub` will generate an error message.

An expression like `y = x + 3` is an assignment statement (unless it is prefaced by an `If`, `ElseIf` or `Do While`—see below). The computer evaluates the right-hand side, by looking up the value already assigned to `x`, adding 3, and storing this value in the memory space reserved for `y`. A statement like `x = x + 3` is perfectly acceptable. It simply adds 3 to the value of `x`. It doesn't matter whether you add spaces around the = and + signs; the VBA editor will automatically adjust the spacing.

It is optional whether you must specifically allocate memory space for a variable. If you type "Option Explicit" in a VBA module, then all variables must be declared. This is done with the keyword `Dim` at the beginning of the program (more on this below). If you do not type "Option Explicit," then you can create a new variable in the middle of a program simply by assigning it a value. For example, you can type `y = x+3`. If `y` has not been previously defined, then it will be created and assigned the value `x+3`. If `x` has not been defined, it will be created and given the value 0.

The main virtue of selecting "Option Explicit" is that it helps to avoid typographical errors. Suppose for example that you intend to assign a new value to a variable named `HardToSpell`. If you misspell the name in the assignment statement and have not declared "Option Explicit," then VBA will create a new variable with the misspelled name. The program will still execute, but it will not calculate what you intended it to calculate. Likewise, if you intend to perform some operation with `HardToSpell` and assign the result to another variable and you misspell `HardToSpell`, then a new variable will be created with the misspelled name, given a value of zero, and the operation will be performed with the value zero rather than with the value of `HardToSpell`. In both cases, with "Option Explicit" declared, VBA will generate an error message alerting you to the misspelling.

A.6 Mathematical Operations

The basic mathematical operations are performed in VBA in the same way as in Excel: addition, subtraction, multiplication (the asterisk symbol), division (/), and exponentiation (the caret symbol—`3^2` is 3 squared). The natural exponential is also the same in VBA as in Excel: `Exp(6)` is e^6. The square root and natural logarithm functions are also available in VBA but with different names than in Excel. The name of the square root function is `Sqr` in VBA (rather than `Sqrt` as in Excel) and the name of the natural logarithm function is `Log` in VBA (rather than `Ln` as in Excel). It does not matter whether or not you capitalize the names; the VBA editor will automatically capitalize, converting for example `exp` to `Exp`.

Other mathematical functions are used in VBA by preceding their Excel names with `Application.` For example `Application.Max(3,4)` returns the larger of 3 and 4. Of course, VBA means "Visual Basic for Applications" and the application being used here is Excel, so the name `Application.Max` indicates that the Excel `Max` function is to be used. A function that we use frequently is `Application.NormSDist(d)`, which returns the probability that a standard normal random variable is less than or equal to `d`.

A.7 Random Numbers

Computers do not behave in a random way (though of course it may seem like it when one crashes) but they can generate sequences of numbers that pass statistical tests for randomness. The basic construction is the generation of a random integer in some range $[0, N]$ with each integer in the range being "equally likely." Dividing by N gives a number between 0 and 1 that has the appearance of being uniformly distributed. This number can then be transformed to give the appearance of a normal distribution or other standard distributions. Random integers are generated sequentially by an algorithm of the type $I_j = aI_{j-1} + c \bmod N$, for constants a and c. "mod N" means the remainder after dividing by N (7 mod 5 is 2, 10 mod 3 is 1, etc.). In this construction, I_{j-1} is called the "seed," and each integer becomes the seed for the next. This is certainly not a random construction, but if the constants and N are suitable chosen (N must be very large) then the integers will have the appearance of unpredictability, both to the human observer and according to formal statistical tests.

VBA has a built-in function for generating random variables that are uniformly distributed between 0 and 1. This function is called `Rnd()`. The same function is in Excel but called `Rand()`. Applying the inverse of the standard normal cumulative distribution function to a random variable that is uniformly distributed between 0 and 1 will generate a random variable with the standard normal distribution (i.e, the normal distribution with mean 0 and

variance 1). The inverse of the standard normal cumulative distribution function is provided in Excel as the `NormSInv` function, and hence it can be called in VBA as `Application.NormsInv`. Given the existence of the `NormSInv` function, this is the simplest, though not the fastest, way to transform a uniformly distributed random variable into a normally distributed one. To reduce typing, the following function is used throughout this book.

```
Function RandN()
   RandN = Application.NormSInv(Rnd())
End Function
```

A.8 For Loops

A loop is a command or set of commands that executes repeatedly either for a fixed number of times or until some condition is violated. To execute the commands for a fixed number of times, use a "for loop."

To add the first 10 integers together we can create the following macro:

```
Sub AddIntegers()
x = 1
For i = 2 To 10
    x = x + i
Next i
ActiveCell.Value = x
End Sub
```

In the above, we first initialized the value of `x` to be 1. The statement(s) between the `For` statement and the `Next` statement are executed repeatedly. In the first passage through the loop, the variable `i` has the value 2 and the statement `x = x + i` translates as `x = 1 + 2`, so `x` is given the value 3. In the next passage, `i` has the value 3 and `x` has the value 3, so the statement `x = x + i` translates as `x = 3 + 3`, and `x` is given the value 6, etc.

Any variable name (not just `i`) can be used as a counter. The indentation of the line `x = x + i` is optional and serves only to make the program easier to read.

The number of iterations need not be fixed when the program is written. We can use variables in the `For` statement like `For i = y To z`. The number of iterations will then be determined by the values of `y` and `z` when the for loop is encountered.

In the statement `For i = 2 To 10`, MATLAB increases `i` by one each time it reaches the statement `Next i`. This is the default, but it can be changed. If you want `i` to increase by two each time, you can write

```
For i = 2 To 10 Step 2.
```

Negative step sizes and non-integer step sizes are also acceptable. For example, the statement `For i = 10 To 1 Step -1` produces a loop that executes "backwards," starting from `i = 10` and counting down until `i = 2`.

A.9 While Loops and Logical Expressions

A "while loop" executes a block of statements repeatedly until some condition is violated. For example a crude way to add the first 10 integers would be with the following macro:

```
Sub AddIntegers2()
x = 0
i = 1
Do While i <= 10
    x = x + i
    i = i + 1
Loop
ActiveCell.Value = x
End Sub
```

When the program first encounters the `Do While` statement, it checks whether the condition `i` ≤ 10 is true. If it is, then the statements preceding the `Loop` statement are executed. The condition `i` ≤ 10 is then checked again, and the statements are executed repeatedly in this way until the condition `i` ≤ 10 is false. Be careful that the statements being executed will eventually cause the condition to be false.

The comparison operators that can be used in the `Do While` statement (and `If` and `ElseIf` described below) are less than (`<`), less than or equal to (`<=`), greater than (`>`), greater than or equal to (`>=`), and equal to (`=`).

The expression `Not(i > 10)` is equivalent to `(i <= 10)`. Multiple conditions can be combined: the expression `i <= 10 And y > 6` is true if (and only if) both `i` ≤ 10 and `y` > 6 are true, and the expression `i <= 10 Or y > 6` is true if either or both of its component statements is true.

A.10 If, Else, and ElseIf Statements

You can cause a statement to execute only when a certain condition is satisfied by prefacing it with an `If` statement. The format is

```
If y <= 10 Then
    x = 2 * x
End If
```

which doubles `x` if `y` ≤ 10 and does nothing otherwise. Rather than doing nothing otherwise, you can cause a different statement or block of statements to execute when the condition is violated by including an `Else`. For example,

```
If y <= 10 Then
    x = 2 * x
Else
    x = 3 * x
End If
```

In this case, if y > 10, the statements following the `Else` statement execute, tripling `x`. Finally, you can check multiple conditions sequentially with `ElseIf`. Consider the following:

```
If y <= 10 Then
    x = 2 * x
ElseIf y <= 20 Then
    x = 3 * x
ElseIf y <= 30 Then
    x = 4 * x
Else
    x = 5 * x
End If
```

The conditions are checked sequentially as follows. If $y \leq 10$, then `x` is doubled and execution of of the `If` block ends. If $y > 10$, the condition $y \leq 20$ is checked. If this is true, `x` is tripled. If it is not true, the next condition is checked, etc. The result is that `x` is doubled when $y \leq 10$; it is tripled when $10 < y \leq 20$; it is quadrupled when $20 < y \leq 30$; and it is quintupled when $y > 30$.

A.11 Variable Declarations

As mentioned before, if "Option Explicit" is declared, each variable must be declared at the beginning of a subroutine or function. A variable can be declared to be of a specific type or the type can be left unspecified and VBA will choose what seems to be the appropriate type. For numerical calculations, the important types are Integer, Long, Double, and Variant. The Integer data type is for storage of integers between -32,768 and 32,769. The Long data type can store integers between plus or minus 2 billion (actually a bit more than 2 billion). The Double data type stores arbitrary (floating point) numbers, to sixteen digits of accuracy. The Variant data type is the default type for variables whose type is not specified, and it adjusts itself automatically to the data stored within it. To declare a variable to be of a particular type, there are two equally acceptable syntaxes. For example, the Double type can be declared either as `Dim x As Double` or `Dim x#`. The Integer type can be declared either as `Dim x As Integer` or `Dim x%`. Note that the syntax `Dim i, j, k As Integer` is acceptable but it declares only `k` as being of type Integer, with `i` and `j` still being of type Variant. On the other hand, `Dim i%, j%, k%` declares `i`, `j` and `k` as being of type Integer.

In this book, the data type is left unspecified (hence as Variant), with the exception that the type of large arrays is declared. The Variant data type requires more memory for storage, so this is a bit inefficient.

Variables declared within a function or subroutine are "local variables." They can only be accessed within the function or subroutine within which they are defined. To understand this, consider the following simple example of a function (`TestFunction`) calling another function (`AddTwo`).

```
Function TestFunction(x)
TestFunction = x * AddTwo(x)
End Function

Function AddTwo(x)
Dim y
y = x + 2
AddTwo = y
End Function
```

The result of `TestFunction(3)` is $3 \times 5 = 15$. Consider now the following (strange) change to `TestFunction`.

```
Function TestFunction(x)
Dim z
z = x * AddTwo(x)
TestFunction = y
End Function
```

The new feature is that `TestFunction(x)` attempts to return `y`, which is defined only in `AddTwo`. If `TestFunction(3)` is executed, then one of two things will happen: (i) if "Option Explicit" has been declared, an error message will appear with the information that the variable `y` has not been declared within `TestFunction`, or (ii) if "Option Explicit" has not been declared, the function will return a value of zero. The reason in both cases is that the variable `y` defined within `AddTwo` is not available to `TestFunction`—it is local to `AddTwo`. In case (ii), a new variable `y` is created within `TestFunction` and, like all new variables, is given a default value of zero. The error message is probably preferable in this circumstance, which points again to the value of the "Option Explicit" declaration.

It is possible to declare a variable so that it is available to (and can be modified by) all of the functions in a module, or all of the functions in a workbook, or even all of the functions in all open workbooks. Such variables are called "global variables." That facility is useful in some situations, but it is not used in this book.

A.12 Variable Passing

As we have seen, functions and macros can call other functions or macros to perform part of their work. For example, macros shown previously in this appendix call the `MsgBox` function. The default arrangement in VBA is that variables are passed to functions (or to macros—though variables are not passed to macros in this book) "by reference" rather than "by value." This means that the actual memory location of the variable is given to the function, and any changes made to a variable by a function will affect the use of the variable in a calling function. Consider, for example the following simple change to the function `AddTwo`:

```
Function AddTwo(x)
x = x + 2
AddTwo = x
End Function
```

This function still adds the number 2 to its input. Now when we execute

```
TestFunction(3)
```

and it reaches the line

```
TestFunction = x * AddTwo(x)
```

it will be multiplying `x` by 5 as before. However, now `x` has been changed in `AddTwo` from 3 to 5, so the result of `TestFunction(3)` is $5 \times 5 = 25$.

This may sometimes be what one wants, but it is more likely that it will produce mistakes. There are two possible solutions. One is to change the function `AddTwo` as follows:

```
Function AddTwo(ByVal x)
x = x + 2
AddTwo = x
End Function
```

This forces VBA to pass only the value of `x` and not the memory location. So when 2 is added to `x` and returned to `TestFunction(3)`, the value of `x` in `TestFunction` is still 3.

The second solution is more straightforward: simply do not change input variables within a function. That is, we can use our first version of `AddTwo`, which created a new variable to store the sum of `x` and 2, rather than changing the value of `x` (or we could use the simpler one-line function `AddTwo = x + 2`). The functions in this book follow this second approach—**we avoid changing the values of input variables**.

A.13 Arrays

It is very useful to be able to use a single variable name to store multiple values. For example, we can write loops such as

```
For i = 1 To 10
    x(i) = whatever
Next i
```

An array variable must be declared, regardless of whether "Option Explicit" is declared. Normally, the declaration takes the form `Dim x(10)` if the largest index number of `x` is known (to equal 10) when the function or macro is written. The default in VBA is that the first element is indexed by 0.[2] Therefore,

[2] This can be changed so that the default is for the first element to be indexed by 1 with the statement "Option Base 1."

`Dim x(10)` creates a vector with 11 elements, which are accessed as `x(0)`, ..., `x(10)`. The type of each element is Variant, unless it is declared otherwise—for example, `Dim x(10) As Integer` reserves memory locations for 11 integers. Multidimensional arrays can also be used. For example, `x(10, 6, 12)` creates a 3-dimensional array, with $11 \times 7 \times 13$ elements. The first index does not have to be zero. The declaration `Dim x(1 To 10)` creates a vector with 10 elements, which are accessed as `x(1)`, ..., `x(10)`. Likewise, one can use, for example `Dim x(-6 To 3)` to start the indexing at -6 and end at 3.

If the dimension of the array is not fixed, which is often the case, then normally it must be declared with empty parentheses—for example, `Dim x()`. The dimension will depend on the input arguments, or on calculations based on the input arguments. Before the array is used, the program must include a statement specifying the dimension, of the form `ReDim x(N)`, or `ReDim x(1 To N)`, where the variable `N` is either an input argument to the function or has been calculated prior to the statement `ReDim x(N)`.

The exception to the above statements about declaring array variables, whether the number of elements is known in advance or not, is when an array is assigned to a variable by a call to a function. The `Array` function is one example of a function that creates an array. For example

```
Dim x
x = Array(3, 6, 7)
```

will create an array with elements `x(0)=3`, `x(1)=6`, and `x(2)=7`. Replacing `Dim x` with `Dim x(2)` in this context will not work.

Functions can take arrays as inputs and return arrays as outputs. Arrays can be input by (i) typing the array as an argument of the function, (ii) inputting the worksheet cells in which the array resides, or (iii) passing the array as an output from another function. An array created in one function is passed to another function in the same way that any other variable is passed. To type an array as an input, enclose it curly braces, separate items in each row with a comma, and separate rows with a semicolon—for example $\{3, 1, 2; 4, 6, 2\}$ is an array with two rows and three columns, the first row being $\{3, 1, 2\}$ and the second row being $\{4, 6, 2\}$. The same array might be input via cell references as `B3:C5`.

Arrays can also be output to Excel worksheets. Consider the following:

```
Function MyArray(x)
Dim y(3)
For i = 1 To 3
  y(i) = i * x
Next i
MyArray = y
End Function
```

Note that the array `y` has four elements. The program does not define element 0, so it is zero by default. If we execute MyArray(2), the other elements will be `y(1) = 2`, `y(2) = 4`, and `y(3) = 6`. If we execute the function by

typing `=MyArray(2)` in a cell of a worksheet, the number 0 will appear. (To avoid this and have the output show up in three cells instead of four, we could have declared `Dim y(1 To 3)`.) To see the rest of the output, highlight the active cell and the three cells immediately to the right on the same row. Click the function key F2 and then hold down the key combination CTRL-SHIFT-ENTER. This is the standard Excel procedure for displaying arrays returned by functions. For example, the output of Excel's matrix algebra functions, such as `MMULT`, is revealed in the same way.[3] Two-dimensional arrays can be output to worksheets in the same way.

A.14 Debugging

Errors (bugs) are inevitable. VBA will catch some types (for example, syntax errors) and inform you. The more troublesome errors are those that do not prevent the program from running but lead to incorrect results. It is essential therefore to debug each program carefully.

To debug a subroutine, put the cursor on the subroutine name in the Visual Basic editor. Click on Debug/Step Into (or the function key F8) to step through the subroutine one line at a time. Putting the cursor over any variable will show the value of the variable at that stage of the program. To observe the values of variables more systematically, you can include statements of the form `Debug.Print x` or `Debug.Print "The value of x is " & x` in the subroutine. The subroutine will then print to the Immediate Window. To view the Immediate Window, click View/Immediate Window. Click on Run/Reset (or the square button on the toolbar) to discontinue debugging.

To debug a function, one can rewrite it as a subroutine, defining values for the input variables in the beginning of the subroutine. The VBA debugger has many other features. Debug/Step Over is particularly useful for stepping over a line that does not need to be checked and will be time consuming to check, for example, a call to another function.

[3] Once this is done, the individual cells in which the array was output cannot be changed. Attempting to do so will generate an error message, and it may be necessary to hit the Escape key once or twice to allow any use of the worksheet after the error message appears.

B

Miscellaneous Facts about Continuous-Time Models

B.1 Girsanov's Theorem

In Sect. 2.9, we were able to compute the expected return of an asset under different numeraires directly, by using Itô's formula and the fact that the ratio of a non-dividend-paying asset price to the numeraire asset price is a martingale under the measure associated with the numeraire. In other cases (e.g., Heston's stochastic volatility model and Vasicek's model) the drift of a process could not be computed directly when we changed numeraires, because the process (volatility in Heston's model and the short rate in Vasicek's model) was not an asset price. In general, the change in the drift of a process when we change numeraires (or, more generally, change probability measures) is given by Girsanov's theorem.

An heuristic explanation of Girsanov's theorem is as follows. Let λ be a constant, and let B be a Brownian motion under a probability measure that we will denote by prob. Let $B^*(t) = B(t) + \lambda t$; i.e., $\mathrm{d}B^* = \mathrm{d}B + \lambda\,\mathrm{d}t$. Girsanov's theorem shows how to change the probability measure so that the drift of B^* is zero, i.e., how to change the probability measure to make B^* a martingale and hence (by Levy's theorem) a Brownian motion.

Consider discrete time periods of length Δt and approximate B by a binomial process that steps up or down by $\sqrt{\Delta t}$ in each time period, with up and down being equally likely. This approximation implies that the changes ΔB of the binomial process have mean equal to zero and variance equal to Δt, just as for a true Brownian motion. We have $\Delta B^* = \lambda\,\Delta t \pm \sqrt{\Delta t}$. If we change the probability of the up move to $(1 - \lambda\sqrt{\Delta t})/2$ and the probability of the down move to $(1 + \lambda\sqrt{\Delta t})/2$, then the expected change in B^* will be

$$\left(\frac{1-\lambda\sqrt{\Delta t}}{2}\right)\left(\lambda\,\Delta t + \sqrt{\Delta t}\right) + \left(\frac{1+\lambda\sqrt{\Delta t}}{2}\right)\left(\lambda\,\Delta t - \sqrt{\Delta t}\right) = 0\,.$$

Therefore, B^* is a martingale under these revised probabilities.

Changing the probabilities of each "branch" of the binomial tree in this way implies that the probability of a path through the tree is changed as follows. The probability of a path is the product of the probabilities of the branches, so, letting prob* denote the revised probabilities, we have

$$\frac{\text{prob}^*(\text{path through time } t)}{\text{prob}\,(\text{path through time } t)} = \frac{\text{prob}^*(\text{path through time } t-\Delta t)}{\text{prob}\,(\text{path through time } t-\Delta t)} \times \frac{\text{prob}^*(\text{branch at } t)}{\text{prob}\,(\text{branch at } t)} \,.$$

Note that our definitions imply

$$\frac{\text{prob}^*(\text{up branch at } t)}{\text{prob}\,(\text{up branch at } t)} = \frac{\frac{1}{2}\left(1-\lambda\sqrt{\Delta t}\right)}{1/2} = 1-\lambda\,\Delta B(t)\,,$$

and

$$\frac{\text{prob}^*(\text{down branch at } t)}{\text{prob}\,(\text{down branch at } t)} = \frac{\frac{1}{2}\left(1+\lambda\sqrt{\Delta t}\right)}{1/2} = 1-\lambda\,\Delta B(t)\,.$$

Therefore,

$$\frac{\text{prob}^*(\text{path through time } t)}{\text{prob}\,(\text{path through time } t)} = \frac{\text{prob}^*(\text{path through time } t-\Delta t)}{\text{prob}\,(\text{path through time } t-\Delta t)} \times \big(1-\lambda\,\Delta B(t)\big)\,.$$

If we let $Y(t)$ denote the ratio of path probabilities through time t, this shows that the percent change in Y at time t is $-\lambda\,\Delta B(t)$, i.e.,

$$Y(t) = Y(t-\Delta t)\times\big(1-\lambda\,\Delta B(t)\big) \Longrightarrow \frac{Y(t)-Y(t-\Delta t)}{Y(t-\Delta t)} = -\lambda\,\Delta B(t)\,.$$

A continuous-time formulation of this equation is

$$\frac{\mathrm{d}Y(t)}{Y(t)} = -\lambda\,\mathrm{d}B(t)\,.$$

This equation implies that Y is a geometric Brownian motion with explicit solution (given that the ratio of path probabilities at date 0 is $Y(0)=1$)

$$Y(t) = \exp\left(-\lambda^2 t/2 - \lambda B(t)\right)\,. \tag{B.1}$$

The above heuristic argument suggests that the process (B.1) defines a ratio of path probabilities, prob* to prob, such that B^* is a martingale under prob*. Because B^* is continuous and its quadratic variation through each

date t is equal to t (because the addition of λt to B does not alter the quadratic variation of B), Levy's theorem implies that B^* must in fact be a Brownian motion relative to the measure prob*. This is the content of Girsanov's theorem. In the formal statement, there is no reference to ratios of path probabilities, because individual paths actually have zero probability under either prob or prob*. Instead, the theorem states that B^* is converted to a Brownian motion by multiplying the probability of any event (set of paths) by the conditional expectation of Y, given the event.

There is no need to assume λ is a constant, provided the random process λ is sufficiently regular that the general form of (B.1), i.e.,

$$Y(t) \equiv \exp\left\{-\frac{1}{2}\int_0^t \lambda^2(u)\,du - \int_0^t \lambda(u)\,\mathrm{d}B(u)\right\}, \tag{B.2}$$

is a martingale.[1]

Girsanov's Theorem: Let B be a Brownian motion on a time horizon $[0,T]$ and let λ be a stochastic process such that Y defined by (B.2) is a martingale. Define

$$B^*(t) = B(t) + \int_0^t \lambda(u)\,du, \tag{B.3}$$

and define a new probability measure prob* by setting prob$^*(A) = 0$ for each event A such that $\mathrm{prob}(A) = 0$, and by defining

$$\mathrm{prob}^*(A) = E\big[Y(T)|A\big] \times \mathrm{prob}(A) \tag{B.4}$$

for each event A such that $\mathrm{prob}(A) > 0$. Then B^* is a Brownian motion on the time horizon $[0,T]$ relative to prob*.

The definition of prob* in the boxed statement emphasizes the ratio of probabilities aspect. It is equivalent to the definition

$$\mathrm{prob}^*(A) = E\left[1_A Y(T)\right] \tag{B.5}$$

for each event A. Thus, it is consistent with the definition (1.11) of the probability of an event A when we use a non-dividend-paying asset price S as the numeraire. The relation between the two is that the "ratio of path probabilities" $Y(T)$ equals $\phi(T)S(T)/S(0)$, where $\phi(T)$ denotes the random state price at date T.

[1] The process (B.2) is an Itô process with zero drift. A sufficient condition for it to be a martingale is that

$$E\left[\exp\left\{\frac{1}{2}\int_0^T \lambda^2(u)\,du\right\}\right] < \infty\,.$$

This is called "Novikov's condition." See, e.g., Karatzas and Shreve [45].

Note also that for any random variable X (for which the mean exists) the mean of X under prob*, which we denote by $E^*[X]$, is given by

$$E^*[X] = E[Y(T)X] \; . \tag{B.6}$$

In some cases we may be given (perhaps by equilibrium arguments) the random variable Y defining the change of measure, and we wish to compute the change in the drift of a Brownian motion (in order to compute, for example, the drift of a volatility or an interest rate). Thus, we need to reverse the above process, in which we started with the change of drift λ and computed Y. This is straightforward. Given $Y(T)$, define $Y(t) = E_t[Y(T)]$, i.e., the expectation of $Y(T)$ under the original measure, given information at date t. Equation (B.2) shows that

$$\frac{\mathrm{d}Y}{Y} = -\lambda \, \mathrm{d}B \; .$$

Therefore,

$$-(\mathrm{d}B)\left(\frac{\mathrm{d}Y}{Y}\right) = \lambda \, \mathrm{d}t \; .$$

It follows that the definition

$$\mathrm{d}B^* = \mathrm{d}B - (\mathrm{d}B)\left(\frac{\mathrm{d}Y}{Y}\right)$$

gives us a Brownian motion B^* relative to the measure prob*. In other words, the drift of B under the measure prob* is $(\mathrm{d}B)(\mathrm{d}Y)/Y$.

B.2 Distribution of the Minimum of a Geometric Brownian Motion

Here we will give an explanation of formulas used in Chap. 8 for valuing barrier and lookback options. From a mathematical point of view, our discussion will be decidedly informal.

Consider an asset price S satisfying

$$\mathrm{d} \log S = \mu \, \mathrm{d}t + \sigma \, \mathrm{d}B \; ,$$

for constants μ and σ, where B is a Brownian motion. Consider constants $K \geq L$ with $L < \log S(0)$. Define $z = \min_{0 \leq t \leq T} S(t)$. Define

$$x = \begin{cases} 1 & \text{if } S(T) > K \text{ and } z > L \; , \\ 0 & \text{otherwise} \; . \end{cases}$$

To price a down-and-out call, we need to compute prob$(x = 1)$. As in Sect. 8.6, define

$$y = \begin{cases} 1 & \text{if } S(T) > K \text{ and } z \le L\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

The event $S(T) > K$ is the union of the disjoint events $x = 1$ and $y = 1$, so we have

$$\begin{aligned} \text{prob}(x = 1) &= \text{prob}(S(T) > K) - \text{ prob}(y = 1) \\ &= \text{N}(d) - \text{prob}(y = 1)\,, \end{aligned}$$

where

$$d = \frac{\log\left(\frac{S(0)}{K}\right) + \mu T}{\sigma\sqrt{T}}\,. \tag{B.7}$$

Thus, the remaining task is to compute $\text{prob}(y = 1)$.

To price lookback options, it is necessary to know the cumulative distribution function of z, i.e., we need to know $\text{prob}(z \le L)$ for arbitrary L. The event $z \le L$ is the union of the disjoint events $S(T) \le L$ and $y = 1$, where we specialize to the case $K = L$ in the definition of y. Thus,

$$\begin{aligned} \text{prob}(z \le L) &= \text{prob}(S(T) \le L) + \text{prob}(y = 1) \\ &= \text{N}(-d) + \text{ prob}(y = 1)\,, \end{aligned}$$

where again we take $K = L$ in the definition of d. Thus, for pricing lookbacks also, the key task is to compute $\text{prob}(y = 1)$.

Assume first that $\mu = 0$, so $\log S$ is a Brownian motion with zero drift. We want to compute the probability of the paths of $\log S$ that dip below $\log L$ and end above $\log K$. Each such path has a "twin" defined by reflecting the path (as in a mirror image) through the horizontal line $x(t) = \log L$ after the first time $\log S$ hits $\log L$. The original path increases by at least $\log K - \log L$ after hitting $\log L$ (otherwise, it could not end above $\log K$). So, the twin decreases by at least $\log K - \log L$ after hitting $\log L$. This means that it ends below $2\log L - \log K$. Moreover, each path ending below $2\log L - \log K$ is the twin in this sense of a path hitting $\log L$ and then ending above $\log K$. Because $\log S$ has no drift, the "twins" are equally likely. Therefore, when $\mu = 0$,

$$\begin{aligned} \text{prob}(y = 1) &= \text{prob}\big(\log S(T) \le 2\log L - \log K\big) \\ &= \text{prob}\left(\frac{B(T)}{\sqrt{T}} \le \frac{2\log L - \log K - \log S(0) - \mu T}{\sigma\sqrt{T}}\right) \\ &= \text{N}(d^*)\,, \end{aligned}$$

where

$$d^* = \frac{\log\left(\frac{L^2}{KS(0)}\right)}{\sigma\sqrt{T}}\,. \tag{B.8}$$

Now consider the case $\mu \neq 0$, the case in which we are really interested. By Girsanov's theorem, the process B^* defined by $B^*(0) = 0$ and

$$\mathrm{d}B^* = \mathrm{d}B + \frac{\mu}{\sigma}\,\mathrm{d}t$$

is a Brownian motion under the measure prob* defined by (B.1) and (B.4), where we take $\lambda = \mu/\sigma$ in the definition of $Y(T)$. The purpose of this definition is that we have

$$\mathrm{d}\log S = \mu\,\mathrm{d}t + \sigma\left(\mathrm{d}B^* - \frac{\mu}{\sigma}\,\mathrm{d}t\right) = \sigma\,\mathrm{d}B^*\ .$$

Letting E denote expectation relative to the measure under which B is a Brownian motion and E^* denote expectation relative to prob*, we have from (B.6) that

$$\begin{aligned}
\mathrm{prob}(y=1) = E[y] &= E\left[Y(T)\frac{y}{Y(T)}\right] \\
&= E^*\left[\frac{y}{Y(T)}\right] \\
&= E^*\left[\exp\left(\frac{1}{2}\lambda^2 T + \lambda B(T)\right) y\right] \\
&= E^*\left[\exp\left(\frac{1}{2}\lambda^2 T + \lambda[B^*(T) - \lambda T]\right) y\right] \\
&= E^*\left[\exp\left(-\frac{1}{2}\lambda^2 T + \lambda B^*(T)\right) y\right]\ . \qquad \text{(B.9)}
\end{aligned}$$

Because $\log S$ has no drift under prob*, the twin paths described before are equally likely under prob*. However, the reflection leads to low values of $\log S(T)$ and hence to low values of $B^*(T)$ rather than high values, and we must compensate for this in (B.9). Specifically, for a path of $\log S$ that ends above $\log K$, we have

$$B^*(T) = \frac{\log K - \log S(0) + \varepsilon}{\sigma} \qquad \text{(B.10)}$$

for some $\varepsilon > 0$ and the reflection of this path has

$$B^*(T) = \frac{2\log L - \log K - \log S(0) - \varepsilon}{\sigma} \qquad \text{(B.10}'\text{)}$$

for the same ε. Therefore, to use the reflected path, we compute

$$\varepsilon = 2\log L - \log K - \log S(0) - \sigma B^*(T)$$

from (B.10$'$) and substitute this into the right-hand side of (B.10) to obtain

$$\begin{aligned}
&\frac{\log K - \log S(0) + 2\log L - \log K - \log S(0) - \sigma B^*(T)}{\sigma} \\
&\qquad = \frac{2\log L - 2\log S(0)}{\sigma} - B^*(T)
\end{aligned}$$

as the value that should replace $B^*(T)$ in (B.9) when we use the reflected paths. As in the case $\mu = 0$, using the reflected paths means replacing the random variable y with y' defined as

$$y' = \begin{cases} 1 & \text{if } \log S(T) \le 2\log L - \log K \ , \\ 0 & \text{otherwise} \ . \end{cases}$$

Substituting into (B.9) and employing some algebra gives us

$$\begin{aligned} \text{prob}(y=1) &= E^* \left[\exp\left(-\frac{1}{2}\lambda^2 T + \lambda \left[\frac{2\log L - 2\log S(0)}{\sigma} - B^*(T) \right] \right) y' \right] \\ &= \left(\frac{L}{S(0)} \right)^{2\mu/\sigma^2} E^* \left[\exp\left(-\frac{1}{2}\lambda^2 T - \lambda [B(T) + \lambda T] \right) y' \right] \\ &= \left(\frac{L}{S(0)} \right)^{2\mu/\sigma^2} E^* \left[\exp\left(-\frac{3}{2}\lambda^2 T - \lambda B(T) \right) y' \right] \\ &= \left(\frac{L}{S(0)} \right)^{2\mu/\sigma^2} E \left[\exp\left(-2\lambda^2 T - 2\lambda B(T) \right) y' \right] \ , \end{aligned} \tag{B.11}$$

where for the last equality we used (B.6) again.

Now we will define another change of measure. Set $\delta = 2\lambda$,

$$Z(T) = \exp\left(-\delta^2 T/2 - \delta B(T) \right)$$

and $\text{prob}^{**}(A) = E[1_A Z(T)]$ for each event A. From the definition of δ and (B.6) we have

$$\begin{aligned} E\left[\exp\left(-2\lambda^2 T - 2\lambda B(T) \right) y' \right] &= E\left[\exp\left(-\frac{1}{2}\delta^2 T - \delta B(T) \right) y' \right] \\ &= E^{**}[y'] \\ &= \text{prob}^{**}(y' = 1) \ . \end{aligned} \tag{B.12}$$

Moreover, Girsanov's theorem states that $\mathrm{d}B^{**} = \mathrm{d}B + \delta\,\mathrm{d}t$ defines a Brownian motion B^{**} under the measure prob^{**}. The event $y' = 1$ is equivalent to

$$\begin{aligned} &\log S(0) + \mu T + \sigma B(T) \le \log\left(\frac{L^2}{K} \right) \\ &\qquad \Longleftrightarrow \log S(0) + \mu T + \sigma [B^{**}(T) - \delta T] \le \log\left(\frac{L^2}{K} \right) \\ &\qquad \Longleftrightarrow \log S(0) - \mu T + \sigma B^{**}(T) \le \log\left(\frac{L^2}{K} \right) \\ &\qquad \Longleftrightarrow \frac{B^{**}(T)}{\sqrt{T}} \le d' \ , \end{aligned} \tag{B.13}$$

where we define

$$d' = \frac{\log\left(\frac{L^2}{KS(0)}\right) + \mu T}{\sigma\sqrt{T}} \,. \tag{B.14}$$

Combining (B.11), (B.12) and (B.13) yields

$$\text{prob}(y = 1) = \left(\frac{L}{S(0)}\right)^{2\mu/\sigma^2} \text{N}(d') \,.$$

Summarizing, we have

Assume $\mathrm{d}\log S = \mu\,\mathrm{d}t + \sigma\,\mathrm{d}B$ where B is a Brownian motion. Define $z = \min_{0\le t\le T} S(t)$. For $K \ge L$ and $L \le \log S(0)$,

1. The probability that $S(T) > K$ and $z > L$ is

$$\text{N}(d) - \left(\frac{L}{S(0)}\right)^{2\mu/\sigma^2} \text{N}(d') \,,$$

   where d is defined in (B.7) and d' is defined in (B.14).
2. The probability that $z \le L$ is

$$\text{N}(-d) + \left(\frac{L}{S(0)}\right)^{2\mu/\sigma^2} \text{N}(d') \,,$$

   where d is defined in (B.7) and d' is defined in (B.14), substituting $K = L$ in both.

B.3 Bessel Squared Processes and the CIR Model

This section will present additional results regarding the CIR square-root short rate process discussed in Sect. 14.4. The ideas described here are one way (though not the only way) to derive the CIR discount bond option pricing formula. We begin with the following simpler process

$$\mathrm{d}x(t) = \delta\,\mathrm{d}t + 2\sqrt{x(t)}\,\mathrm{d}Z \tag{B.15}$$

for a Brownian motion Z and constant $\delta > 0$. This is called a Bessel-squared process with parameter δ. The parameter δ determines whether x can ever reach zero. If $\delta \ge 2$, then with probability one, $x(t)$ is strictly positive for all t; whereas, if $\delta < 2$, then with positive probability, x will sometimes hit zero (but will never go negative).

In the particular (rare) case that δ is an integer, the squared length of a δ-dimensional vector of independent Brownian motions is a process x satisfying (B.15). To see this, let $B_1, \ldots, B_\delta$ be independent Brownian motions starting at given values b_i; i.e., $B_i(0) = b_i$. Define $x(t) = \sum_{i=1}^{\delta} B_i(t)^2$. Then Itô's formula gives us

$$\begin{aligned} \mathrm{d}x(t) &= \sum_{i=1}^{\delta} 2B_i(t)\,\mathrm{d}B_i(t) + \sum_{i=1}^{\delta} \mathrm{d}t \\ &= \delta\,\mathrm{d}t + 2\sqrt{x(t)} \sum_{i=1}^{\delta} \frac{B_i(t)}{\sqrt{x(t)}}\,\mathrm{d}B_i(t). \end{aligned}$$

The process Z defined by $Z(0) = 0$ and

$$\mathrm{d}Z = \sum_{i=1}^{\delta} \frac{B_i(t)}{\sqrt{x(t)}}\,\mathrm{d}B_i(t)$$

is a Brownian motion (because it is a continuous martingale with $(\mathrm{d}Z)^2 = \mathrm{d}t$); thus, we obtain (B.15).

Continuing to assume that δ is an integer and that $x(t) = \sum_{i=1}^{\delta} B_i(t)^2$, note that, for any t, the random variables ξ_i defined as $\xi_i = [B_i(t) - B_i(0)]/\sqrt{t}$ are independent standard normals, and we have

$$\begin{aligned} x(t) &= \sum_{i=1}^{\delta} \big[b_i + B_i(t) - B_i(0)\big]^2 \\ &= t \times \sum_{i=1}^{\delta} \left(\frac{b_i}{\sqrt{t}} + \xi_i\right)^2. \end{aligned}$$

A random variable of the form $\sum_{i=1}^{\delta} (\gamma_i + \xi_i)^2$, where the γ_i are constants and the ξ_i are independent standard normals, is said to have a non-central chi-square distribution with δ degrees of freedom and noncentrality parameter $\sum_{i=1}^{\delta} \gamma_i^2$. Thus, $x(t)$ is equal to t times a non-central chi-square random variable with δ degrees of freedom and noncentrality parameter

$$\sum_{i=1}^{\delta} \frac{b_i^2}{t} = \frac{x(0)}{t}\ .$$

The noncentral chi-square distribution can be defined for a non-integer degrees of freedom also, and a process x satisfying (B.15) for a non-integer δ has the same relation to it, namely,

> If x satisfies (B.15), then for any t and $\alpha > 0$, the probability that $x(t) \le \alpha$ is equal to the probability that $z \le \alpha/t$, where z is a random variable with a non-central chi-square distribution with δ degrees of freedom and noncentrality parameter $x(0)/t$.

Now consider the CIR process (14.9). Define $\delta = 4\kappa\theta/\sigma^2$ and define x by (B.15), with $x(0) = r(0)$. Set[2]

$$h(t) = \frac{\sigma^2}{4\kappa}\left(\mathrm{e}^{\kappa t} - 1\right) ,$$

and

$$r(t) = \mathrm{e}^{-\kappa t} x(h(t)) .$$

Then it can be shown[3] that r satisfies the CIR equation (14.9), namely

$$\mathrm{d}r = \kappa(\theta - r)\,\mathrm{d}t + \sigma\sqrt{r}\,\mathrm{d}B \tag{B.16}$$

for a Brownian motion B. For any t and $\alpha > 0$, the probability that $r(t) \le \alpha$ is equal to the probability that $x(h(t)) \le \mathrm{e}^{\kappa t}\alpha$. In view of the previous boxed statement, this implies:

> If r satisfies the CIR equation (B.16) where κ, θ and σ are positive constants, then, for any $t > 0$ and any α, the probability that $r(t) \le \alpha$ is the probability that $z \le \mathrm{e}^{\kappa t}\alpha/h(t)$, where z is a random variable with a non-central chi-square distribution with $\delta = 4\kappa\theta/\sigma^2$ degrees of freedom and noncentrality parameter $r(0)/h(t)$.

To derive the discount bond option pricing formula for the CIR model, we need to know the distribution of $r(T)$ when the parameters κ and θ are time-dependent. Let w denote either u (the maturity of the underlying) or T (the maturity of the option). Using the discount bond maturing at w as the numeraire, we repeat here (14.23), dropping now the "hat" on $\hat{r}$:

$$\mathrm{d}r(t) = \kappa^*(t)[\theta^*(t) - r(t)]\,\mathrm{d}t + \sigma\sqrt{r(t)}\,\mathrm{d}B^*(t) , \tag{B.17}$$

where

$$\kappa^*(t) = \kappa + \sigma^2 b(w - t) \qquad \text{and} \qquad \theta^*(t) = \frac{\kappa\theta}{\kappa^*(t)} .$$

Because $\kappa^*(t)\theta^*(t) = \kappa\theta$, we again define $\delta = 4\kappa\theta/\sigma^2$ but now set

$$h^*(t) = \frac{\sigma^2}{4}\int_0^t \exp\left(\int_0^s \kappa^*(y)\,\mathrm{d}y\right)\mathrm{d}s$$

[2] I learned this transformation from unpublished lecture notes of Hans Buehlmann.

[3] The key to this calculation is the fact that if Z is a Brownian motion and h is a continuously differentiable function with $h'(s) > 0$ for all $s > 0$ then B defined by

$$B(t) = \int_0^t \frac{1}{\sqrt{h'(s)}}\,\mathrm{d}Z_{h(s)}$$

is a Brownian motion also.

and

$$r(t) = \exp\left(-\int_0^t \kappa^*(s)\,\mathrm{d}s\right) x(h^*(t)) \ .$$

Then it can be shown that r satisfies (B.17) for a Brownian motion B^*. Thus, as in the previous paragraphs, the probability that $r(T) \le \alpha$, where r satisfies (B.17), is the probability that

$$z \le \frac{\exp\left(\int_0^T \kappa^*(s)\,\mathrm{d}s\right)\alpha}{h^*(T)} \ ,$$

where z has a non-central chi-square distribution with δ degrees of freedom and noncentrality parameter $r(0)/h^*(T)$.

Straightforward calculations, using in particular the fact that $b(\tau) = a'(\tau)/(\kappa\theta)$ and

$$\int \frac{\mathrm{e}^{\gamma t}}{c(t)^2}\,\mathrm{d}t = -\frac{1}{(\kappa+\gamma)\gamma}\int \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{c(t)}\right)\mathrm{d}t = -\frac{1}{(\kappa+\gamma)\gamma c(t)}$$

give us:

$$\exp\left(\int_0^T \kappa^*(s)\,\mathrm{d}s\right) = \frac{\mathrm{e}^{-\gamma T}c(w)^2}{c(w-T)^2}$$

and

$$h^*(T) = \frac{\sigma^2\mathrm{e}^{-\gamma w}c(w)}{4(\kappa+\gamma)\gamma}\left[\frac{c(w)}{c(w-T)} - 1\right] \ ,$$

where γ and c are defined in (14.14). This simplifies somewhat in the case $w = T$ because $c(0) = 2\gamma$. Thus, the probabilities in the CIR option pricing formula (14.21), which are the probabilities of the event shown in (14.22), are as follows:

- $\mathrm{prob}^u\big(P(T,u) > K\big)$ is the probability that

$$z \le -\frac{\mu_u}{\lambda_u}\left(\frac{\int_T^u \phi(s)\,\mathrm{d}s + a(u-T) + \log K}{b(u-T)}\right) \ ,$$

  where z has a non-central chi-square distribution with $4\kappa\theta/\sigma^2$ degrees of freedom and noncentrality parameter $r(0)/\lambda_u$, and

$$\mu_u = \frac{\mathrm{e}^{-\gamma T}c(u)^2}{c(u-T)^2} \ ,$$
$$\lambda_u = \frac{\sigma^2\mathrm{e}^{-\gamma u}c(u)}{4(\kappa+\gamma)\gamma}\left[\frac{c(u)}{c(u-T)} - 1\right] \ .$$

- $\text{prob}^T\big(P(T,u) > K\big)$ is the probability that

$$z \le -\frac{\mu_T}{\lambda_T}\left(\frac{\int_T^u \phi(s)\,\mathrm{d}s + a(u-T) + \log K}{b(u-T)}\right) ,$$

where z has a non-central chi-square distribution with $4\kappa\theta/\sigma^2$ degrees of freedom and noncentrality parameter $r(0)/\lambda_T$, and

$$\mu_T = \frac{\mathrm{e}^{-\gamma T} c(T)^2}{4\gamma^2} ,$$
$$\lambda_T = \frac{\sigma^2 \mathrm{e}^{-\gamma T} c(T)}{4(\kappa+\gamma)\gamma}\left[\frac{c(T)}{2\gamma} - 1\right] .$$

List of Programs

List of Symbols

e, exp	natural exponential
cov	covariance
log	natural logarithm function
max	maximum function
min	minimum function
n	normal density function
prob, prob*	probability
prob^R, prob^S, ...	probability using an asset as numeraire
var	variance
N	cumulative normal distribution function
M	bivariate normal distribution function
a, b, c	constants or functions of time
d	real number, or, as subscript, indicator of down state
d_1, d_2	real numbers
f, g, h, Σ	functions
i, j, k, ℓ, n	integers
p	probability
q	dividend yield (probability in Chap 1)
r	risk-free rate or continuously-compounded return
r_f	foreign risk-free rate
s, t, w, T	real numbers representing time
u	real number, or, as subscript, indicator of up state
v	squared volatility
x	real number, function of time, or random variable
y, z, ξ, ε	real numbers or random variables
A	set of states of the world, or a constant
1_A	random variable that equals 1 when the event A occurs and zero otherwise
B, B^*	Brownian motion
C	call option price

E, E^*	expectation
E_t, E_t^*	conditional expectation at date t
E^R, E^S, ...	expectation using an asset as numeraire
E_t^R, E_t^S, ...	conditional expectation at date t using an asset as numeraire
F	forward price
F^*	futures price
K	exercise price
L	boundary for a barrier option
M, N	integers
P	discount bond or put option price
P^{mkt}	market discount bond price
R	risk-free accumulation factor
$\mathcal{R}$	simple interest rate (LIBOR)
S	asset price
V, W	portfolio values
X, Y, Z	random processes
α, β, κ, μ, ν, θ	constants or functions of time
δ, Γ, Θ, $\mathcal{V}$	option Greeks (delta, gamma, theta, vega)
λ	constant, function of time, or random process
ϕ	constant or function of time (or, in Chap 1, a state price density)
π	irrational number pi (or, in Chap 1, a state price)
ρ	correlation (or option derivative with respect to r)
σ	volatility
τ	time to maturity
Δt	length of time period
ΔB	change in B over a discrete time period
$\sqrt{\ }$	square root
$\times$	multiplication
!	factorial
d	differential
∂	partial differential
$\int$	integral
$\sum$	sum
$\prod$	product
$\Longrightarrow$	implies
$\Longleftrightarrow$	is equivalent to

References

1. Arrow, K.J.: The role of securities in the optimal allocation of risk bearing. Review of Economic Studies, **31**, 91–96 (1964). Translation of Le rôle des valeurs boursières pour la repartition la meillure des risques. Econometrie (1952)
2. Bielecki, T. , Rutkowski, M.: Credit Risk: Modeling, Valuation and Hedging. Springer, Berlin Heidelberg New York (2002)
3. Black, F.: The pricing of commodity contracts. Journal of Financial Economics, **3**, 167–179 (1976)
4. Black, F., Derman, E., Toy, W.: A one-factor model of interest rates and its application to Treasury bond options. Financial Analysts Journal, January/February, 33–39 (1990)
5. Black, F., Karasinski, P.: Bond and option pricing when short rates are lognormal. Financial Analysts Journal, July-August, 52–59 (1991)
6. Black, F., Scholes, M.: The pricing of options and corporate liabilities. Journal of Political Economy, **81**, 637–654 (1973)
7. Bollerslev, T.: Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics, **31**, 307–327 (1986)
8. Boyle, P.: Options: a Monte Carlo approach. Journal of Financial Economics, **4**, 323–338 (1977)
9. Brace, A., Gatarek, D. , Musiela, M.: The market model of interest rate dynamics. Mathematical Finance, **7**, 127–154 (1996)
10. Brandimarte, P.: Numerical Methods in Finance: A MATLAB-Based Introduction, Wiley, New York (2002)
11. Brennan, M., Schwartz, E.: Finite difference methods and jump processes arising in the pricing of contingent claims: a synthesis. Journal of Financial and Quantitative Analysis 13, 461–474 (1978)
12. Brigo, D., Mercurio, F.: Interest Rate Models, Theory and Practice, Springer, Berlin Heidelberg New York (2001)
13. Broadie, M., Detemple, J.: American option valuation: new bounds, approximations, and a comparison of existing methods. Review of Financial Studies, **9**, 1211–1250 (1997)
14. Broadie, M., Glasserman, P.: Estimating security price derivatives using simulation. Management Science, **42**, 269–285 (1996)

15. Broadie, M., Glasserman, P.: Pricing American-style securities using simulation. Journal of Economic Dynamics and Control, **21**, 1323–1352 (1997)
16. Broadie, M., Kaya, O.: Exact simulation of stochastic volatility and other affine jump diffusion processes. Operations Research (forthcoming)
17. Clewlow, L., Strickland, C.: Implementing Derivatives Models, Wiley, New York (1998)
18. Conze, A., Viswanathan.: Path dependent options: the case of lookback options. Journal of Finance, **46**, 1893–1907 (1991)
19. Cox, J., Ingersoll, J., Ross, S.: A theory of the term structure of interest rates. Econometrica, **53**, 385–408 (1985)
20. Cox, J., Ross, S.: The valuation of options for alternative stochastic processes. Journal of Financial Economics, **3**, 145–166 (1976)
21. Cox, J., Ross, S., Rubinstein, M.: Option pricing: a simplified approach. Journal of Financial Economics, **7**, 229–263 (1979)
22. Dai, Q., Singleton, K.: Specification analysis of affine term structure models. Journal of Finance, **55**, 1943–1978 (2000)
23. Drezner, Z.: Computation of the bivariate normal integral. Mathematics of Computation, **32**, 277-279 (1978)
24. Duffie, D., Kan, R.: A yield-factor model of interest rates., Mathematical Finance, **6**, 379–406 (1996)
25. Duffie, D., Singleton, K.: Credit Risk: Pricing, Measurement and Management, Princeton University Press, Princeton, New Jersey (2003)
26. Epps, T. W.: Pricing Derivative Securities, World Scientific Publishing, Singapore (2000)
27. Geman, H., El Karoui, N., Rochet, J.-C.: Changes of numeraire, changes of probability measure and option pricing. Journal of Applied Probability, **32**, 443–458 (1995)
28. Geske, R.: The valuation of compound options. Journal of Financial Economics, **7**, 63–81 (1979)
29. Glasserman, P.: Monte Carlo Methods in Financial Engineering, Springer, New York Berlin Heidelberg (2004)
30. Goldman, M., Sosin, H., Gatto, M.: Path dependent options: 'buy at the low, sell at the high.' Journal of Finance, **34**, 1111–1127 (1979)
31. Harrison, J.M., Kreps, D.: Martingales and arbitrage in multiperiod securities markets. Journal of Economic Theory, **20**, 381–408 (1979)
32. Haug, E.G.: The Complete Guide to Option Pricing Formulas, McGraw-Hill, New York (1998)
33. Heath, D., Jarrow, R., Morton, A.: Bond pricing and the term structure of interest rates: a new methodology for contingent claims valuation. Econometrica, **60**, 77–105 (1992)
34. Heston, S.: A closed-form solution for options with stochastic volatility with applications to bond and currency options. Review of Financial Studies, **6**, 327–344 (1993)
35. Heston, S., Nandi, S.: A closed-form GARCH option valuation model. Review of Financial Studies, **13**, 585–625 (2000)
36. Ho, T., Lee, S.: Term structure movements and pricing interest rate contingent claims. Journal of Finance, **41**, 1011–1029 (1986)
37. Hull, J.: Options, Futures, and Other Derivatives, Prentice-Hall, 5th ed, Upper Saddle River, New Jersey (2002)

38. Hull, J., White, A.: Pricing interest-rate-derivative securities. Review of Financial Studies, **3**, 573–592 (1990)
39. Jäckel, P.: Monte Carlo Methods in Finance, Wiley, New York (2002)
40. Jackson, M., Staunton, M.: Advanced Modelling in Finance Using Excel and VBA, Wiley, New York (2001)
41. James J., Webber, N.: Interest Rate Modelling, Wiley, New York (2000)
42. Jamshidian, F.: An exact bond option formula., Journal of Finance, **44**, 205–209 (1989)
43. Jamshidian, F.: LIBOR and swap market models and measures. Finance and Stochastics, **1**, 293–330 (1997)
44. Jarrow, R., Rudd, A.: Option Pricing, Dow Jones-Irwin, Homewood, Illinois (1983)
45. Karatzas, I., Shreve, S.: Brownian Motion and Stochastic Calculus, Springer, New York Berlin Heidelberg (1988)
46. Leisen, D.P.J., Reimer, M.: Binomial models for option valuation—examining and improving convergence., Applied Mathematical Finance, **3**, 319–346 (1996)
47. Longstaff, F., Schwartz, E.: Interest rate volatility and the term structure: a two-factor general equilibrium model., Journal of Finance, **47**, 1259–1282 (1992)
48. Longstaff, F., Schwartz, E.: Valuing American options by simulation: a simple least-squares approach. Review of Financial Studies, **14**, 113–147 (2001)
49. McDonald, R.: Derivatives Markets, Addison Wesley, Boston (2003)
50. Margrabe, W.: The value of an option to exchange one asset for another. Journal of Finance, **33**, 177–186 (1978)
51. Merton, R.: Theory of rational option pricing. Bell Journal of Economics and Management Science, **4**, 141–183 (1973)
52. Miltersen, K.R., Sandermann, K., Sondermann, D.: Closed form solutions for term structure derivatives with log-normal interest rates., Journal of Finance, **52**, 409–430 (1997)
53. Mina, J., Xiao, J.: Return to RiskMetrics, The Evolution of a Standard (2001)
54. Musiela, M., Rutkowski, M.: Martingale Methods in Financial Modeling, Springer, Berlin Heidelberg New York (1997)
55. Rebonato, R.: Interest-Rate Option Models, 2nd ed., Wiley, New York, (1998)
56. Rebonato, R.: Modern Pricing of Interest-Rate Derivatives, The LIBOR Market Model and Beyond, Princeton University Press, Princeton, New Jersey (2002)
57. Schönbucher, P.: Credit Derivatives Pricing Models: Models, Pricing and Implementation, Wiley, New York (2003)
58. Stulz, R.: Options on the minimum or maximum of two risky assets: analysis and applications, Journal of Financial Economics, **10**, 161–185 (1982)
59. Tavakoli, J.: Credit Derivatives and Synthetic Structures, Wiley, New York (2001)
60. Tavella, D.A.: Quantitative Methods in Derivatives Pricing, Wiley, New York (2002)

61. Trigeorgis, A.: A log-transformed binomial analysis method for valuing complex multi-option investments. Journal of Financial and Quantitative Analysis, **26**, 309–326 (1991)
62. Vasicek, O.: An equilibrium characterization of the term structure. Journal of Financial Economics, **5**, 177–188 (1977)
63. Wilmott, P.: Paul Wilmott on Quantitative Finance, Vol. 2, Wiley, New York (2000)
64. Wilmott, P., Dewynne, J., Howison, S.: Option Pricing: Mathematical Models and Computation, Oxford Financial Press, Oxford (2000)
65. Zhang, P.G.: Exotic Options, A Guide to Second Generation Options, 2nd ed., World Scientific Publishing, Singapore (1998)

Index