Mathematical Methods in Finance

Lecture notes (part II)

Nuno M. Brites

2024/2025

Contents

Preface

This module aims to review basic concepts about mathematical models in finance in order to enable students from various backgrounds (exact sciences and social sciences) to apply techniques in mathematical finance. It is also expected to provide students with a conceptual understanding of the models and techniques used in financial mathematics. Finally, students are expected to acquire knowledge of option pricing models.

All errors and omissions are entirely my responsibility. Please notify me if you find any errors or typos. Suggestions and feedback are also welcome. Have a happy and successful year!

2024 | Nuno M. Brites | nbrites@iseg.ulisboa.pt

Chapter 1

R and RStudio

1.1 About R and RStudio

- R is a free software for statistical (but not only!) computing and graphics.
- R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes:
  - an effective data handling and storage facility;
  - a suite of operators for calculations on arrays, in particular matrices;
  - a large, coherent, integrated collection of intermediate tools for data analysis;
  - graphical facilities for data analysis and display either on-screen or on hardcopy;
  - well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

1.2 Installing R and RStudio

- Go to https://cran.r-project.org/ - this site will offer you all necessary information on R and its packages.
- Download R for Windows or MacOS to install the base distribution.
- Install R.
- Go to https://rstudio.com/products/rstudio/ - this site will offer you all necessary information on RStudio.
- Download RStudio for Windows or MacOS to install the desktop distribution.
- Install RStudio.

1.3 The prompt

- R has a command line interface, and will accept simple commands to it. This is marked by a > symbol, called the **prompt**. If you type a command and press return, R will evaluate it and print the result for you.
- Try now! Write

```
> print("My first R command!")
## [1] "My first R command!"
```

- Lines starting with # are ignored by R and can be used to insert comments in the script.

1.4 Assignments

- The expression x <- 1 creates a **variable** called x and assigns the value 1 to x. Note that the variable on the left is assigned to the value on the right. The left hand side must contain only a single variable name.

- To get the "<-" write the "<" sign and the "-" sign: "x <- 1".
- It is possible (actually is a good idea) to leave spaces between the variable name and its value, but it is not possible to have a space between the $<$ and $-$ signs!
- One can also assign with = (or ->). However, in order to avoid confusion, it is common to use <- to distinguish from the equality operator =.

1.5 Rules for defining variables

- Names of variables can be chosen quite freely in R. They can be built from letters, digits, the period (dot) symbol and the underscore symbol (_).
- However, one should pay attention to:
  - do not start a name with a digit or a period followed by a digit;
  - R is case sensitive, so "A" and "a" refer to different variables;
  - be consistent with variables names throughout the program;
  - avoid names that provide no description, e.g., single-letter names, unless if they are parameters;
  - some names are already used by the system, e.g., FALSE, TRUE, exp, sum, etc.

Chapter 2

R basics

2.1 Basic operations

```
> 2 + 2
## [1] 4
>
> 7 * 17
## [1] 119
>
> sqrt(9)
## [1] 3
>
> 3^3
## [1] 27
>
> log(7)
## [1] 1.946
>
> log10(7)
## [1] 0.8451
```

2.2 Precision

```
> sin(pi/2)
## [1] 1
>
> pi
## [1] 3.142
>
> options(digits = 22)
>
> pi
## [1] 3.141592653589793115998
>
> options(digits = 4)
```

2.3 Infinity or not defined, and missings

```
> 1/0
## [1] Inf
>
> 2 * Inf
```

```
## [1] Inf
>
> -1/0
## [1] -Inf
>
> 0/0
## [1] NaN
>
> c(1, 2, 3, NA, 5)
## [1]  1  2  3 NA  5
>
> mean(c(1, 2, 3, NA, 5))
## [1] NA
>
> mean(c(1, 2, 3, 5))
## [1] 2.75
```

2.4 Important commands

```
> ls()  #...display list of variables in memory
>
> ls.str()  #...display the structure os the list of variables in memory
>
> rm(a)  #...remove an object
>
> rm(list = ls())  #...remove ALL objects
>
```

2.5 Internal help function

```
> help(mean)  # shorthand for help(mean)
```

```
> example(mean)  # examples
```

```
> help.search("mean")  # full information
```

```
> help.start()  # manuals
```

Chapter 3

Data structures

In R we have objects which are functions and objects which are data.

- Function examples:
  - `sin()`
  - `integrate()`
  - `plot()`
  - `paste()`
- Data examples:
  - 42
  - 1:5
  - "R"
  - matrix(1:12, nrow=4, ncol=3)
  - data.frame(a=1:5, tmt=c("a","b","a","b","a"))
  - list(x=2, y="abc", x=1:10)

3.1 Vector

```
> # Vector of numbers, e.g:
> c(1, 1.2, pi, exp(1))
## [1] 1.000 1.200 3.142 2.718
>
> # We can have vectors of other things too, e.g:
> c(TRUE, 1 == 2)
## [1]  TRUE FALSE
> c("a", "ab", "abc")
## [1] "a"   "ab"  "abc"
>
> # But not combinations, e.g:
> c("a", 5, 1 == 2)
## [1] "a"     "5"     "FALSE"
> # Notice that R just turned everything into characters!
```

3.1.1 Constructing vectors

```
> # Integers from 9 to 17
> x <- 9:17
> x
## [1]  9 10 11 12 13 14 15 16 17
>
> # A sequence of 11 numbers from 0 to 1
> y <- seq(0, 1, length = 11)
> y
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
>
> # The same number or the same vector several times
> z <- rep(1:2, 5)
> z
##  [1] 1 2 1 2 1 2 1 2 1 2
>
> # Combine numbers, vectors or both into a new vector
> xz10 <- c(x, z, 10)
> xz10
##  [1]  9 10 11 12 13 14 15 16 17  1  2  1  2  1  2  1  2  1  2 10
```

3.1.2 Index and logical index

```
> # Define a vector with integers from (-5) to 5 and extract the numbers with
> # absolute value less than 3:
> x <- (-5):5
> x
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
>
> # by their index in the vector:
> x[4:8]
## [1] -2 -1  0  1  2
>
> # or, by negative selection (set a minus in front of the indices we don't
> # want):
> x[-c(1:3, 9:11)]
## [1] -2 -1  0  1  2
>
> # A logical vector can be defined by:
> index <- abs(x) < 3
> index
##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
>
> # Now this vector can be used to extract the wanted numbers:
> x[index]
## [1] -2 -1  0  1  2
```

3.2 Matrix

- Similar to vectors we can have matrices of objects of the same type, e.g:

```
> matrix(c(1, 2, 3, 4, 5, 6) + pi, nrow = 2)
##       [,1]  [,2]  [,3]
## [1,] 4.142 6.142 8.142
## [2,] 5.142 7.142 9.142
>
> matrix(c(1, 2, 3, 4, 5, 6) + pi, nrow = 2) < 6
##      [,1]  [,2]  [,3]
## [1,] TRUE FALSE FALSE
## [2,] TRUE FALSE FALSE
>
> # We can create higher order arrays, e.g:
> array(c(1:24), dim = c(4, 3, 2))
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24
```

3.2.1 Constructing matrices

```
>
> # Combine rows into a matrix
> A <- rbind(1:3, c(1, 1, 2))
> A
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    1    2
>
> # Or columns
> B <- cbind(1:3, c(1, 1, 2))
> B
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    1
## [3,]    3    2
>
> # Define a matrix from one long vector
> C <- matrix(c(1, 0, 0, 1, 1, 0, 1, 1, 1), nrow = 3, ncol = 3)
> C
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    0    1    1
## [3,]    0    0    1
>
> # Can also be done by rows by adding 'byrow=TRUE' before the last parenthesis.
> # Try!
```

3.2.2 Index and logical index

```
> A <- matrix((-4):5, nrow = 2, ncol = 5)
> A
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   -4   -2    0    2    4
## [2,]   -3   -1    1    3    5
>
>
> # Negative values
> A[A < 0]
## [1] -4 -3 -2 -1
>
> # Assignments
> A[A < 0] <- 0
> A
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    2    4
## [2,]    0    0    1    3    5
>
> # Matrix rows can be selected by
```

```
> A[2, ]
## [1] 0 0 1 3 5
>
> # and similarly for columns
> A[, c(2, 4)]
##      [,1] [,2]
## [1,]    0    2
## [2,]    0    3
```

3.2.3 Properties of vectors and matrices

- The R function `mode()` when applied to a vector or to a matrix detects the type of singles that is stored:

```
> A <- matrix(rep(c(TRUE, FALSE), 2), nrow = 2)
>
> B <- rnorm(4)
>
> C <- matrix(LETTERS[1:9], nrow = 3)
>
> A
##       [,1]  [,2]
## [1,]  TRUE  TRUE
## [2,] FALSE FALSE
> B
## [1]  0.3831 -1.0932 -1.9698 -1.0487
> C
##      [,1] [,2] [,3]
## [1,] "A"  "D"  "G"
## [2,] "B"  "E"  "H"
## [3,] "C"  "F"  "I"
>
> mode(A)
## [1] "logical"
> mode(B)
## [1] "numeric"
> mode(C)
## [1] "character"
```

- Vectors and matrices have lengths: the length is the number of elements:

```
> x <- matrix(c(NA, 2:12), ncol = 3)
> x
##      [,1] [,2] [,3]
## [1,]   NA    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
>
> length(x[1, ])
## [1] 3
>
> length(x)
## [1] 12
>
> # The dimension of a matrix is the number of rows and columns: The number of
> # columns is the second element:
> dim(x)
## [1] 4 3
> dim(x)[2]
## [1] 3
```

3.2.4 Naming rows and columns in a matrix

- We can add names to a matrix with the `colnames()` and `rownames()` functions:

```
> x <- matrix(rnorm(12), nrow = 4)
> x
##          [,1]     [,2]     [,3]
## [1,]  0.1498  1.57437  0.6664
## [2,] -2.1298  0.53188 -1.6020
## [3,]  2.1301 -1.02010 -0.1869
## [4,]  1.6175 -0.03406 -0.0503
>
> colnames(x) <- paste("data", 1:3, sep = "")
>
> rownames(x) <- paste("obs", 1:4, sep = "")
>
> x
##         data1    data2   data3
## obs1  0.1498  1.57437  0.6664
## obs2 -2.1298  0.53188 -1.6020
## obs3  2.1301 -1.02010 -0.1869
## obs4  1.6175 -0.03406 -0.0503
>
> y <- matrix(rnorm(15), nrow = 5)
> y
##           [,1]     [,2]     [,3]
## [1,]  0.68643  0.70217 -1.1306
## [2,]  0.27738 -3.17149 -1.2519
## [3,] -0.88305 -1.34622 -0.6305
## [4,]  0.99013 -1.02663 -0.4722
## [5,] -0.04314  0.04514  1.3876
>
> colnames(y) <- LETTERS[1:ncol(y)]
>
> rownames(y) <- letters[1:nrow(y)]
>
> y
##           A        B       C
## a  0.68643  0.70217 -1.1306
## b  0.27738 -3.17149 -1.2519
## c -0.88305 -1.34622 -0.6305
## d  0.99013 -1.02663 -0.4722
## e -0.04314  0.04514  1.3876
```

3.2.5 Matrix multiplication

```
> M <- matrix(rnorm(20), nrow = 4, ncol = 5)
> N <- matrix(rnorm(15), nrow = 5, ncol = 3)
>
> M %*% N
##          [,1]     [,2]     [,3]
## [1,] -2.4846 -0.9892  0.08665
## [2,]  4.7616  8.0682 -1.77667
## [3,] -0.3881 -3.8442  0.70953
## [4,] -0.9089  1.6877 -0.46688
>
> # Can we perform N*M? No! A and B are not compatible!! Try to run: N%*%M
```

3.2.6 Additional functions

```
> M <- matrix(rnorm(16), nrow = 4, ncol = 4)
>
> dim(M)
## [1] 4 4
>
> t(M)
##         [,1]    [,2]    [,3]    [,4]
## [1,] -1.362  0.6772 -0.4276 -0.4977
## [2,]  1.287 -0.8655  1.8981  0.2111
## [3,]  1.354 -0.1442  1.4469  1.0165
## [4,]  1.772  1.0606  0.6391 -0.9341
>
> det(M)
## [1] -4.958
>
> (invM <- solve(M))
##           [,1]    [,2]      [,3]    [,4]
## [1,] -0.64117  0.5818  0.708019 -0.0714
## [2,] -0.19141 -0.4558  0.508096 -0.5330
## [3,] -0.04741  0.6431  0.236212  0.8019
## [4,]  0.24677  0.2869 -0.005358 -0.2803
>
> eigen(M)
## eigen() decomposition
## $values
## [1]  1.7629+0.0000i -1.5468+0.0000i -0.9653+0.9416i -0.9653-0.9416i
##
## $vectors
##             [,1]         [,2]             [,3]             [,4]
## [1,] -0.5399+0i  0.68439+0i  0.3422+0.2917i  0.3422-0.2917i
## [2,] -0.1818+0i -0.56630+0i  0.6574+0.0000i  0.6574+0.0000i
## [3,] -0.7936+0i  0.45913+0i -0.3061-0.1670i -0.3061+0.1670i
## [4,] -0.2137+0i -0.01077+0i -0.3221+0.3747i -0.3221-0.3747i
```

3.3 Simple Functions

- The above functions are built-in functions. However, it is simple to write your own functions:

```
> # a square function
> square <- function(x) {
+     x * x
+ }
>
> square(2)
## [1] 4
> square(1:5)
## [1]  1  4  9 16 25
>
> # a power function with two arguments
> power <- function(x, pow) {
+     x^pow
+ }
>
> power(2, 3)
## [1] 8
> power(1:5, 3)
## [1]   1   8  27  64 125
```

3.4 Default arguments

- A function can have default arguments:

```
> power <- function(x, pow = 2) {
+     x^pow
+ }
>
> power(1:5)
## [1]  1  4  9 16 25
>
> power(1:5, 4)
## [1]   1  16  81 256 625
```

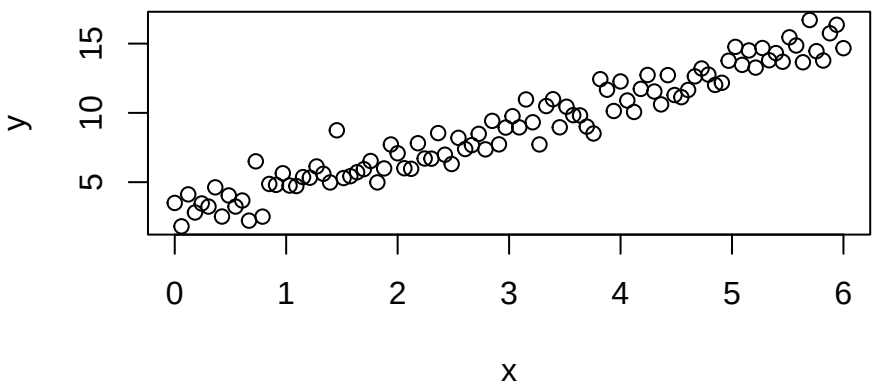
3.5 Integrals and derivatives

```
> f <- function(x) (x^2) * exp(x)
> integrate(f, 0, 1)
## 0.7183 with absolute error < 8e-15
>
> # check: str(integrate(f,0,1)).
```

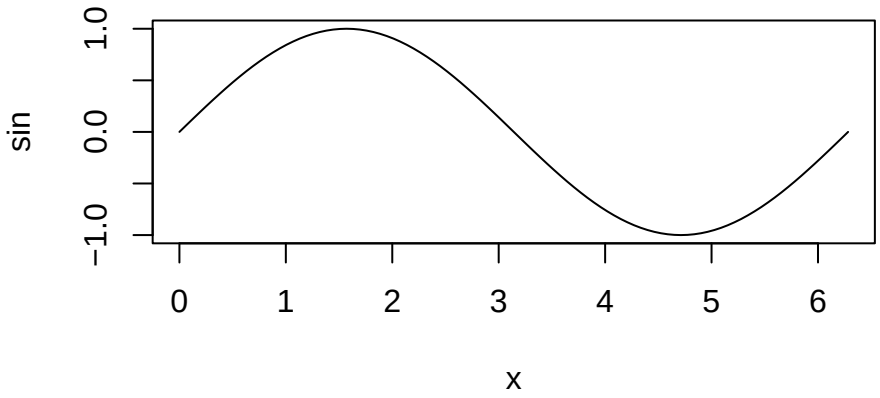
```
> f <- expression((x^2) * exp(x))
> (df <- D(f, "x"))
## 2 * x * exp(x) + (x^2) * exp(x)
>
> x <- c(0, 3, 10)
> eval(df)
## [1]       0.0     301.3 2643175.9
```

3.6 Simple plots

```
> x <- seq(0, 6, length = 100)
>
> y <- 2 * x + 3 + rnorm(100)
>
> plot(x, y)
```

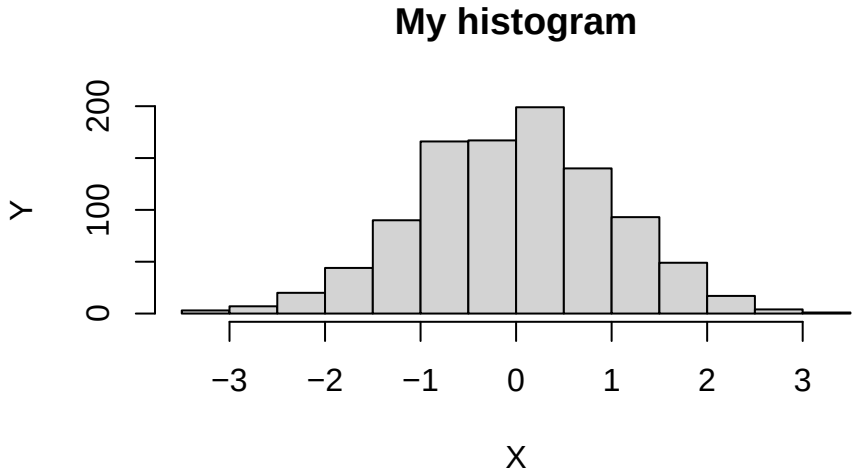



```
>
> plot(sin, 0, 2 * pi)
```

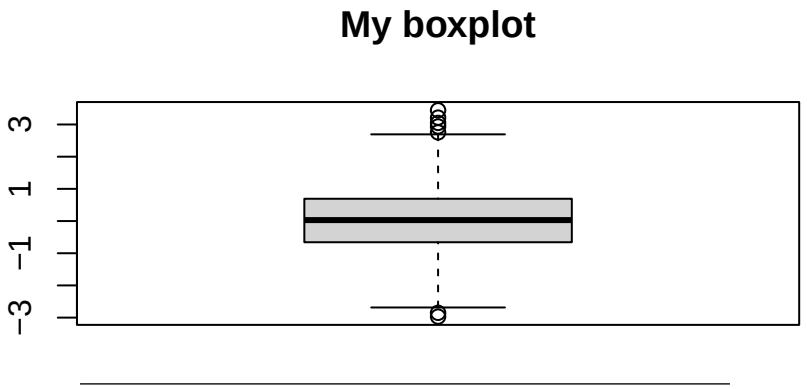


3.6.1 Histogram and boxplot

```
> hist(rnorm(1000), main = "My histogram", xlab = "X", ylab = "Y")
```




```
> boxplot(rnorm(1000), main = "My boxplot")
```



Chapter 4

Loops and flow control

4.1 Loops with `for`

- In R the for loop is used perform a task for each element in a set. The syntax is:

```
> for (var in num1:num2) {
+     instruction1
+     instruction2
+     ...
+ }
```

- Example:

```
> y <- sample(1:5)
> z <- c()
> for (i in 1:5) {
+     z[i] <- y[i]^2
+ }
> z
## [1]  1  4  9 16 25
```

- Another example:

```
> M <- matrix(sample(1:12), 4, 3)
> M
##      [,1] [,2] [,3]
## [1,]    9    5   12
## [2,]   11    4    2
## [3,]    7    3   10
## [4,]    6    1    8
>
> for (i in 1:4) {
+     for (j in 1:3) {
+         print(paste("M(", i, ",", j, ") =", M[i, j]))
+     }
+ }
## [1] "M( 1 , 1 ) = 9"
## [1] "M( 1 , 2 ) = 5"
## [1] "M( 1 , 3 ) = 12"
## [1] "M( 2 , 1 ) = 11"
## [1] "M( 2 , 2 ) = 4"
## [1] "M( 2 , 3 ) = 2"
## [1] "M( 3 , 1 ) = 7"
## [1] "M( 3 , 2 ) = 3"
## [1] "M( 3 , 3 ) = 10"
## [1] "M( 4 , 1 ) = 6"
## [1] "M( 4 , 2 ) = 1"
```

```
## [1] "M( 4 , 3 ) = 8"
```

4.2 Flow control: if and if else statements

The syntax is:

```
if (condition){
  expression_if_true else expression_if_false
}
```

- Example:

```
> (x <- sample(1:3))
## [1] 1 3 2
>
> if (x[1] < x[2] & x[2] < x[3]) print("Ordered vector") else print(sort(x))
## [1] 1 2 3
```

```
> (x <- 1:3)
## [1] 1 2 3
>
> if (x[1] < x[2] & x[2] < x[3]) print("Ordered vector") else print(sort(x))
## [1] "Ordered vector"
```

- Another example:

```
> for (i in 1:3) {
+     if (i == 2)
+         print("The index is 2") else print("The index is not 2")
+ }
## [1] "The index is not 2"
## [1] "The index is 2"
## [1] "The index is not 2"
```

- A shorter version...

```
> ifelse(condition, expression_if_true, expression_if_false)
```

```
> (x <- rnorm(2, 0, 1))
## [1]  0.453 -1.510
> ifelse(x[1] < x[2], "True", "False")
## [1] "False"
```

4.3 While loop example

- The while loop: `while(cond) expr`

```
> x <- 1
> while (x < 5) {
+     x <- x + 1
+     print(x)
+ }
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
> x <- 1
> while (x < 5) {
+     print(x)
+     x <- x + 1
+ }
## [1] 1
```

```
## [1] 2
## [1] 3
## [1] 4
```

Chapter 5

Simulation

- Random numbers can be used for:
  - simulation of complex systems
  - encryption
  - bootstrap methods (the bootstrap method is a resampling technique used to estimate statistics on a population by sampling a dataset with replacement. It can be used to estimate summary statistics such as the mean or standard deviation $\Longrightarrow$ Risk models course)
  - design of experiments
  - performance testing
  - simulation testing of models and estimation
  - ...
- How do we get random numbers?
  1. Use a natural phenomena. E.g: a dice, a deck of cards, ...
  2. Use a table of random numbers.
  3. Computer algorithm: easy, but the algorithm should be:
     - carefully selected
     - fast
     - simple
     - reproducible

Item 3 is included in R. It is easy to simulate from most distributions of interest!

- Random numbers (in R) are actually called pseudo-random numbers (because the algorithm can repeat the sequence, and the numbers are thus not entirely random).
- Generated from an algorithm.
- For all practical purposes, pseudo-random numbers behave like true random numbers.
- By specifying the input to the algorithm, pseudo-random numbers can be re-created.
- The default pseudo random number generator in R is the Mersenne Twister.
- In R the Mersenne Twister uses a random seed as input: constructed from time and session ID.
- You can replace the random seed with a fixed seed with the `set.seed()` function.

5.1 The `set.seed()` function

- Sets of random numbers:

```
> set.seed(12345)
> rnorm(3)
## [1]  0.5855  0.7095 -0.1093
> rnorm(5)
## [1] -0.4535  0.6059 -1.8180  0.6301 -0.2762
> rnorm(3)
## [1] -0.2842 -0.9193 -0.1162
```

5.2 Random numbers and probability distributions

Very useful link.

- Simulation from a uniform distribution:

```
> runif(5, min = 1, max = 2)
## [1] 1.965 1.707 1.645 1.390 1.699
```

- Simulation from a normal distribution:

```
> rnorm(5, mean = 2, sd = 1)
## [1]  2.1107  1.9613 -0.5129  2.4728  1.6459
```

- Simulation from a gamma distribution:

```
> rgamma(5, shape = 2, rate = 1)
## [1] 3.5252 0.4502 3.8128 0.8147 0.1360
```

- Simulation from a binomial distribution:

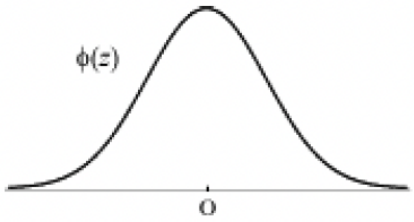
```
> rbinom(5, size = 100, prob = 0.3)
## [1] 34 35 35 26 19
```

- List of functions:

```
> # Density -> d
> dnorm(x, mean = 0, sd = 1, log = FALSE)
>
> # Distribution function -> p
> pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
>
> # Quantile function -> q
> qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
>
> # Random generator -> r
> rnorm(n, mean = 0, sd = 1)
```

- Remember...

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$$



z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
.0	.3989	.3989	.3989	.3988	.3986	.3984	.3982	.3980	.3977	.3973
.1	.3970	.3965	.3961	.3956	.3951	.3945	.3939	.3932	.3925	.3918
.2	.3910	.3902	.3894	.3885	.3876	.3867	.3857	.3847	.3836	.3825
.3	.3814	.3802	.3790	.3778	.3765	.3752	.3739	.3725	.3712	.3697

Figure 5.1: Standard Gaussian distribution.

- Distributions:

```
>
> # normal: dnorm(x=..., mean=..., sd=...); x is a quantile.
>
> # dnorm(0,0,1) = 1/sqrt(2*pi)
> dnorm(0, 0, 1)
```

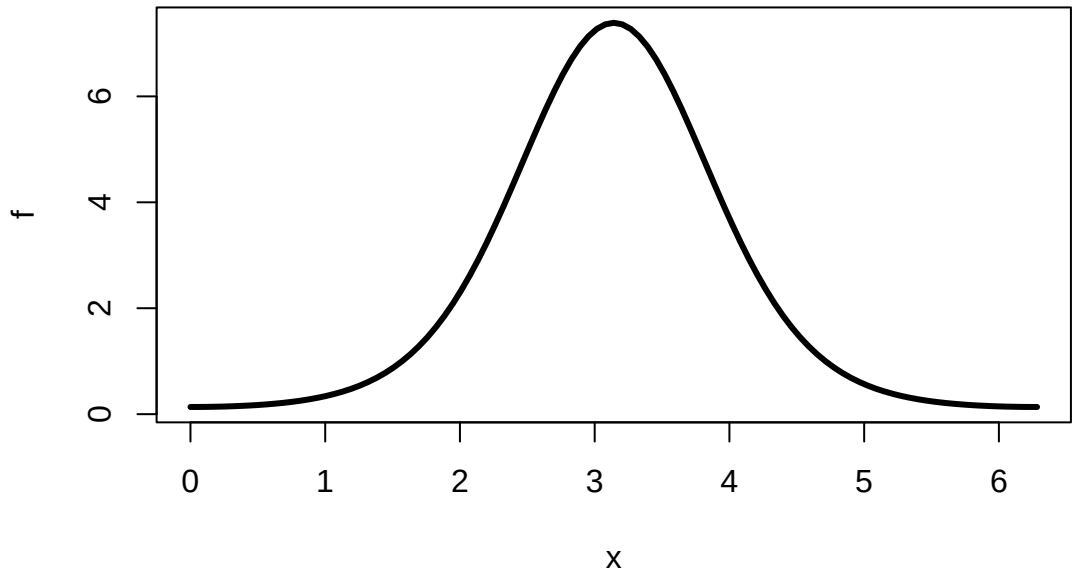
```
## [1] 0.3989
> dnorm(0, 0, 1) == 1/sqrt(2 * pi)
## [1] TRUE
>
> # normal: pnorm(q=..., mean=..., sd=...); q is a quantile.
> pnorm(0, 0, 1)
## [1] 0.5
> pnorm(-1.96, 0, 1)
## [1] 0.025
> pnorm(-1.64, 0, 1)
## [1] 0.0505
> pnorm(1.96, 0, 1)
## [1] 0.975
> pnorm(1.64, 0, 1)
## [1] 0.9495
>
> # normal: qnorm(p=..., mean=..., sd=...); p is a probability.
> qnorm(0.5, 0, 1)
## [1] 0
> qnorm(0.95, 0, 1)
## [1] 1.645
> qnorm(0.975, 0, 1)
## [1] 1.96
```

5.3 Monte Carlo integration (example)

- Consider the function $f(x) = e^{2\cos(x-\pi)}$ and compute the integral

$$\int_0^{2\pi} f(x)dx :$$

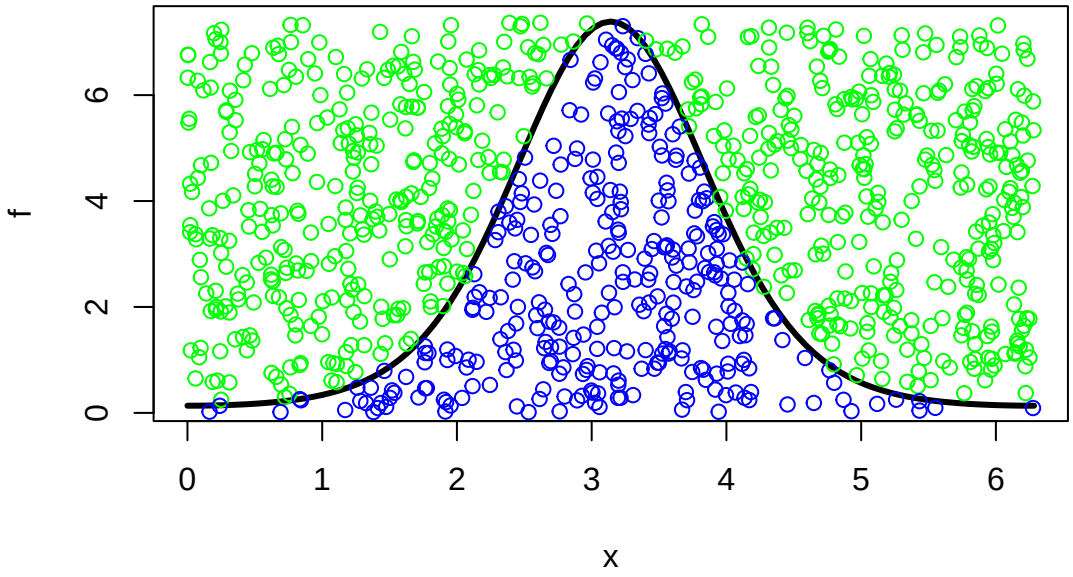
```
> rm(list = ls())
> f <- function(x) exp(2 * cos(x - pi))
> plot(f, 0, 2 * pi, lwd = 3)
```



- Monte Carlo method:
  - simulate uniformly distributed points $(x_1, y_1), \ldots, (x_n, y_n)$ in $[0, 2\pi] \times [0, 8]$ within a "box". Note that $0 < f(x) < e^2$, $\forall\, x \in [0, 2\pi]$:

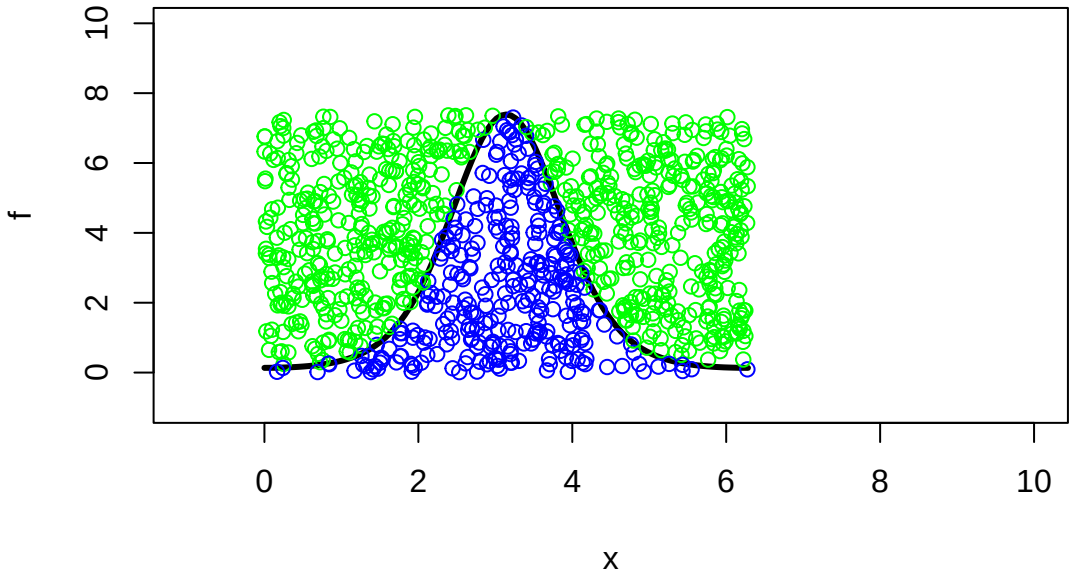
```
> set.seed(1234)
> plot(f, 0, 2 * pi, lwd = 3)
> x <- runif(1000, 0, 2 * pi)
```

```
> y <- runif(1000, 0, exp(2))
> mycol <- ifelse(y < f(x), "blue", "green")
> points(x, y, col = mycol)
```



(to get a wider view of what's happening:)

```
> set.seed(1234)
> plot(f, 0, 2 * pi, xlim = c(-1, 10), ylim = c(-1, 10), lwd = 3)
> x <- runif(1000, 0, 2 * pi)
> y <- runif(1000, 0, exp(2))
> mycol <- ifelse(y < f(x), "blue", "green")
> points(x, y, col = mycol)
```



- estimate the probability of points (say m) being below the curve of f: m/n.
- the area of the "box" is $2\pi \times e^2 \approx 46.43$, so the estimated integral, i.e., the area under the curve, becomes

$$\int_0^{2\pi} f(x)dx \approx 46.43 \times \frac{m}{n}.$$

```
> set.seed(4321)
>
> x <- runif(1e+06, 0, 2 * pi)
> y <- runif(1e+06, 0, exp(2))
> m.over.n <- sum(y < f(x))/length(x)
>
```

```
> options(digits = 20)
> (MCint <- m.over.n * 46.43)
## [1] 14.325558629999999738
>
> # 'real' value
> (Realint <- integrate(f, 0, 2 * pi)$value)
## [1] 14.323056878100514311
>
> (MCint - Realint)
## [1] 0.0025017518994854270886
>
> options(digits = 4)
```

Chapter 6

Graphics in R (exploratory learning)

6.1 base

- The original (default) graphics system in R.
- Highly customizable.
- Complex plots require many code lines.

```
> demo(graphics)
```

```
> demo(persp)
```

- All the code. Try it!

```
> library(datasets)
> library(grDevices)
> library(graphics)
> library(gridGraphics)
>
>
> # demo1
> x <- stats::rnorm(50)
> par(bg = "white")
> plot(x, ann = FALSE, type = "n")
> abline(h = 0, col = gray(0.9))
> lines(x, col = "green4", lty = "dotted")
> points(x, bg = "limegreen", pch = 21)
> title(main = "Simple Use of Color In a Plot", xlab = "Just a Whisper of a Label",
+     col.main = "blue", col.lab = gray(0.8), cex.main = 1.2, cex.lab = 1, font.main = 4,
+     font.lab = 3)
>
> # demo2
> par(bg = "gray")
> pie(rep(1, 24), col = rainbow(24), radius = 0.9)
> title(main = "A Sample Color Wheel", cex.main = 1.4, font.main = 3)
> title(xlab = "(Use this as a test of monitor linearity)", cex.lab = 0.8, font.lab = 3)
>
> # demo3
> pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
> names(pie.sales) <- c("Blueberry", "Cherry", "Apple", "Boston Cream", "Other", "Vanilla Cream")
> pie(pie.sales, col = c("purple", "violetred1", "green3", "cornsilk", "cyan", "white"))
> title(main = "January Pie Sales", cex.main = 1.8, font.main = 1)
> title(xlab = "(Don't try this at home kids)", cex.lab = 0.8, font.lab = 3)
>
> # demo4
> par(bg = "cornsilk")
> n <- 10
> g <- gl(n, 100, n * 100)
```

```
> x <- rnorm(n * 100) + sqrt(as.numeric(g))
> boxplot(split(x, g), col = "lavender", notch = TRUE)
> title(main = "Notched Boxplots", xlab = "Group", font.main = 4, font.lab = 1)
>
> # demo5
> par(bg = "white")
> n <- 100
> x <- c(0, cumsum(rnorm(n)))
> y <- c(0, cumsum(rnorm(n)))
> xx <- c(0:n, n:0)
> yy <- c(x, rev(y))
> plot(xx, yy, type = "n", xlab = "Time", ylab = "Distance")
> polygon(xx, yy, col = "gray")
> title("Distance Between Brownian Motions")
>
>
> # demo6
> x <- c(0, 0.4, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11, 1.73, 2.05, 2.02)
> par(bg = "lightgray")
> plot(x, type = "n", axes = FALSE, ann = FALSE)
> usr <- par("usr")
> rect(usr[1], usr[3], usr[2], usr[4], col = "cornsilk", border = "black")
> lines(x, col = "blue")
> points(x, pch = 21, bg = "lightcyan", cex = 1.25)
> axis(2, col.axis = "blue", las = 1)
> axis(1, at = 1:12, lab = month.abb, col.axis = "blue")
> box()
> title(main = "The Level of Interest in R", font.main = 4, col.main = "red")
> title(xlab = "1996", col.lab = "red")
>
> # demo7
> par(bg = "cornsilk")
> set.seed(1)
> x <- rnorm(1000)
> hist(x, xlim = range(-4, 4, x), col = "lavender", main = "")
> title(main = "1000 Normal Random Variates", font.main = 3)
>
> # demo8
> pairs(iris[1:4], main = "Edgar Anderson's Iris Data", font.main = 4, pch = 19)
>
> # demo9
> pairs(iris[1:4], main = "Edgar Anderson's Iris Data", pch = 21, bg = c("red", "green3",
+     "blue")[unclass(iris$Species)])
>
> # demo10
> x <- 10 * 1:nrow(volcano)
> y <- 10 * 1:ncol(volcano)
> lev <- pretty(range(volcano), 10)
> par(bg = "lightcyan")
> pin <- par("pin")
> xdelta <- diff(range(x))
> ydelta <- diff(range(y))
> xscale <- pin[1]/xdelta
> yscale <- pin[2]/ydelta
> scale <- min(xscale, yscale)
> xadd <- 0.5 * (pin[1]/scale - xdelta)
> yadd <- 0.5 * (pin[2]/scale - ydelta)
> plot(numeric(0), numeric(0), xlim = range(x) + c(-1, 1) * xadd, ylim = range(y) +
+     c(-1, 1) * yadd, type = "n", ann = FALSE)
> usr <- par("usr")
```

```
> rect(usr[1], usr[3], usr[2], usr[4], col = "green3")
> clines <- contourLines(x, y, volcano, levels = lev)
> lapply(clines, lines, col = "yellow", lty = "solid")
> box()
> title("A Topographic Map of Maunga Whau", font = 4)
> title(xlab = "Meters North", ylab = "Meters West", font = 3)
> mtext("10 Meter Contour Spacing", side = 3, line = 0.35, outer = FALSE, at = mean(par("usr")[1:2]),
+     cex = 0.7, font = 3)
>
> # demo11
> par(bg = "cornsilk")
> coplot(lat ~ long | depth, data = quakes, pch = 21, bg = "green3")
```

Chapter 7

Mathematical Methods in Finance

7.1 Introduction

A **derivative** is a **contract** which promises to make a **payment** at a specified time in the future, the amount of which depends upon the behavior of some **underlying asset** (or market variable) up to and including the **time** of the payment. Examples:

- Options on stocks: the underlying asset is the price of stock shares.
- Futures contract on wheat: the underlying asset is the price of wheat.
- Forward contract for selling US dollars: the underlying asset is the exchange rate EUR-USD.

Derivatives may depend on many other variables: interest rates, indexes, electricity price, etc. Other derivatives are credit derivatives, insurance derivatives, weather derivatives, etc...

Definition 7.1. We will work with European options. Remember some definitions on Options:

- **Call option** ("call"): A call option gives the holder the right (but not the obligation) to **buy** the underlying asset by a certain date T for a certain price K.
- **Put option** ("put"): A put option gives the holder the right (but not the obligation) to **sell** the underlying asset by a certain date T for a certain price K.
- The price K in the contract is known as the exercise price or **strike** price.
- The date T in the contract is known as the expiration date or **maturity**.
- European options can be exercised only on the expiration date itself.
- American options can be exercised at any time up to the expiration date.
- Long position: investor that buys the option.
- Short position: investor that sells the option (writer of the option).
- Long position on a "call": Payoff $= \max\{S_T - K, 0\}$.
- Short position on a "call": Payoff $= -\max\{S_T - K, 0\} = \min\{K - S_T, 0\}$.
- Long position on a "put": Payoff $= \max\{K - S_T, 0\}$.
- Short position on a "put": Payoff $= -\max\{K - S_T, 0\} = \min\{S_T - K, 0\}$.

Remark. Notation:

- t is the current time.
- S_t is the underlying share price at time t.
- K is the strike or exercise price.
- T is the option expiry date.
- C_t is the price at time t of European call options.
- P_t is the price at time t of European put options.
- c_t is the price at time t of American call options.
- p_t is the price at time t of American put options.
- r is the risk-free rate of interest (assumed constant).

Remark. Terminology:

- A call option is:
  - in-the-money if $S_t > K$.
  - out-of-money if $S_t < K$.
  - at-the-money if $S_t = K$.
- A put option is:
  - in-the-money if $S_t < K$.
  - out-of-money if $S_t > K$.
  - at-the-money if $S_t = K$.

7.2 The binomial model

- This section introduces the binomial tree model and uses it to **compute arbitrage-free prices for European-style options**.
- Although the mathematical description of this model involves only simple algebra, it is a powerful tool to understand arbitrage pricing theory.
- The main idea of the binomial model is to break the time to maturity of an option into periods.
- Then, in each period, and given the underlying asset's price at the beginning of the period, it assumes that the stock price will change to one of two possible values at the end of the period.
- Thus, we can then determine the **value of the option** recursively by starting at the maturity date, evaluating the option's value under each possibility for the final prices of the stock, and then moving backward through the (binomial) tree.
- The binomial model is a model in **discrete time** and it leads to the celebrated Black-Scholes model (a **continuous time model**) as a limiting case.
- Basic assumptions of the binomial model: (1) no trading costs or taxes; (2) no minimum or maximum units of trading; (3) stocks and bounds can only be traded at discrete times ($t = 1, 2, 3, \ldots$); (4) the principle of no arbitrage holds.

7.2.1 One-period binomial model

We start with the one-period version of the model. The model assumes that we have two assets: a risk-free asset (e.g., a treasury bond, currency, ...) and a stock. Here, we denote by B_t and S_t the bond and stock prices at time t, respectively. Typically, time $t = 0$ represents the present time, and $t = 1$ denotes some future time.

Let us begin by describing the behavior of the stock prices. If at time $t = 0$ the price is S_0, then the stock price at time $t = 1$, S_1, is given by the following random variable

$$S_1 = \begin{cases} uS_0\,, & \text{with probability } p_u \text{ (i.e., if the price goes up)} \\ dS_0\,, & \text{with probability } p_d \text{ (i.e., if the price goes down)}\,, \end{cases}$$

where $u > 1$, $d < 1$, p_u, and p_d are positive positive constants satisfying $d < u$, and $p_u + p_d = 1$. Figure 7.1 shows the development of the price of a stock under the above specification.

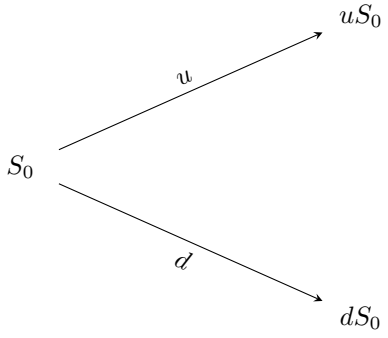


Figure 7.1: Stock prices in a one-period binomial tree.

It is often convenient to write instead

$$S_1 = S_0 Z,$$

where Z is a random variable defined as

$$Z = \begin{cases} u\,, & \text{with probability } p_u\,, \\ d\,, & \text{with probability } p_d\,. \end{cases}$$

On the other hand, the bond price is deterministic and given by

$$\begin{aligned} B_0 &= 1\,, \\ B_1 &= 1 + R\,, \end{aligned}$$

where R is the interest rate for the period.

Remark. We may consider a bond with a continuous interest rate, i.e.,

$$B_1 = e^R, \quad R > 0.$$

Remember: $e^x \approx 1 + x$.

Portfolios and arbitrage

We will consider portfolios constituted of risky assets S and bonds B. More specifically, we will denote by x the number of bonds we hold in our portfolio, y the number of stock units held, and let $h = (x, y)$. Consider now a fixed portfolio $h = (x, y)$. Then, the value of the portfolio at time t, V_t^h, is given by

$$V_t^h = xB_t + yS_t\,.$$

Note that this portfolio has a deterministic value at $t = 0$, $V_0^h = xB_0 + yS_0$, and a stochastic value at $t = 1$, $V_1^h = xB_1 + yS_1 = xB_1 + yS_0 Z$, where Z is a r.v.

We now introduce one of the central concepts of the theory in this section: arbitrage. An arbitrage opportunity is a strategy where we can make a sure profit with no risk ("a free lunch"). Generally speaking, an arbitrage strategy can be created by:

- Start at time zero with a portfolio that has a net value of zero (implying that we are long in some assets and short in others);
- At some future time T:

$$P(\text{"}loss\text{"}) = 0 \quad \text{and} \quad P(\text{"}positive\ profit\text{"}) > 0.$$

.

- If an arbitrage opportunity exists, all the market participants would exploit it, and the market prices of the assets in the portfolio would quickly change to remove the arbitrage opportunity. Formally,

Definition 7.2. An **arbitrage** portfolio is a portfolio h with the properties

$$\begin{aligned} V_0^h &= 0\,, \\ V_1^h &> 0 \quad \text{with probability } 1\,. \end{aligned}$$

Here we will consider the **Principle of No Arbitrage**: arbitrage strategies do not exist. If there are no arbitrage than any two portfolios that give exactly the same payments must have the same price ("Law of One Price").

Essentially an arbitrage portfolio is a deterministic money-making machine. Hence, we can interpret the existence of an arbitrage portfolio as a severe case of mispricing on the market. Therefore, it is natural to investigate when a given market model is arbitrage-free.

Example 7.1. Suppose a stock is listed on both the NYSE and NASDAQ stock exchanges and consider the following steps:

- A trader observes that the current price of the stock on the NYSE is \$10.10 and that on the NASDAQ it is \$10.20.
- They purchase 10,000 of the lower-priced shares (on the NYSE), costing \$101,000 and simultaneously sell the same quantity of 10,000 higher-priced shares, costing \$102,000.

- They manage to pocket the difference (102,000-101,000 = $1000) as profit (assuming there is no commissions).
- Effectively, arbitrage is risk-free profit. At the end of the two transactions (if executed successfully), the trader is not holding any stock position (so they are risk-free), yet they have made a profit!

It turns out that our binomial model above is free of arbitrage if and only if the following conditions hold:

$$d \leq 1 + R \leq u\,.$$

Remark: or, in continuous time, $d < e^R < u$.

Moreover, the above no arbitrage condition is equivalent to saying that $1 + R$ is a convex combination of u and d, i.e.,

$$1 + R = uq_u + dq_d\,,$$

where $q_u, q_d \geq 0$ and $q_u + q_d = 1$. In particular, the weights q_u and q_d can be interpreted as probabilities for a "new probability measure Q" with the property $Q(Z = u) = q_u$, $Q(Z = d) = q_d$. Denoting the expectation w.r.t. this measure by $\mathbb{E}^Q$, it is easy to see that

$$S_0 = \frac{1}{1+R}\mathbb{E}^Q[S_1]\,.$$

Q is known as the "**risk-neutral measure**". Furthermore, q_u and q_d are explicit and given by

$$q_u = \frac{(1+R) - d}{u - d}\,,$$
$$q_d = \frac{u - (1+R)}{u - d}\,.$$

These are known as the **risk-neutral probabilities** and will play an essential role in option pricing.

The above motivates the concept of **risk-neutral valuation**: the price of an asset is the expectation at time 1, calculated using the risk-neutral probabilities, and then discounted using the risk-free rate of interest.

Exercise 7.1. Obtain q_u and q_d.

Risk-neutral pricing

Definition 7.3. A **contingent claim** (eg: a financial derivative such as an option) is any random variable X of the form $X = \Phi(S_1)$, where Φ is some given real valued function known as the *contract function*. The interpretation is that the contract holder **receives the stochastic amount** X **at time** $t = 1$.

Two important examples are the European call and put options. For an European call option with strike K, we have that the $X = \max(S_1 - K, 0)$, while for the European put option with same strike K, the claim is $X = \max(K - S_1, 0)$. In what follows, we will denote by $\Phi(u)$ and $\Phi(d)$, the evaluations $\Phi(uS_0)$ and $\Phi(dS_0)$, respectively.

Our main problem is now determining the "fair" price for a given contingent claim X. We denote the price of X at time t by $\Pi(t; X)$. For a one-period model one have $\Pi(1; X) = X$, i.e. the contract holder receives the stochastic amount X at time $t = 1$.

However, the hard part of the problem is determining $\Pi(0; X)$. The way to solve this problem is to find a portfolio h such that

$$V_1^h = X\,.$$

We call to h a **hedging** portfolio or a **replicating** portfolio (because it replicates the payoff at time 1 on the derivative without any risk). The price at $t = 0$ of the claim X is given by

$$\Pi(0; X) = V_0^h\,.$$

Let us now find the replicating portfolio $h = (x, y)$ for a contingent claim X with contract function Φ. The value of the portfolio at time $t = 1$ is

$$V_1^h = X = \Phi(S_1) = \Phi(uS_0) = \Phi(u) \quad \text{if} \quad Z = u\,,$$
$$V_1^h = X = \Phi(S_1) = \Phi(dS_0) = \Phi(d) \quad \text{if} \quad Z = d\,.$$

Substituting the expression for the value of the portfolio, we obtain the following system of equations:

$$V_1^h = \Phi(u) \iff xB_1 + yuS_0 = \Phi(u) \iff x(1+R) + yuS_0 = \Phi(u)\,,$$
$$V_1^h = \Phi(d) \iff xB_1 + ydS_0 = \Phi(d) \iff x(1+R) + ydS_0 = \Phi(d)\,.$$

Solving for x and y in the above system gives

$$x = \frac{1}{1+R}\frac{u\Phi(d) - d\Phi(u)}{u-d}\,,$$
$$y = \frac{1}{S_0}\frac{\Phi(u) - \Phi(d)}{u-d}\,.$$

Then, the price at $t = 0$ of the claim X, $\Pi(0; X)$, can be written as

$$\begin{aligned}\Pi(0;X) &= V_0^h \\ &= xB_0 + yS_0 \\ &= x + yS_0 \\ &= \frac{1}{1+R}\left(\frac{(1+R)-d}{u-d}\Phi(u) + \frac{u-(1+R)}{u-d}\Phi(d)\right)\,.\end{aligned}$$

Here, we recognize the risk-neutral probabilities q_u and q_d, and we can rewrite the pricing formula above as

$$\Pi(0;X) = \frac{1}{1+R}\left(\Phi(u)q_u + \Phi(d)q_d\right)\,.$$

Thus, the right-hand side can now be interpreted as an expected value under the risk-neutral probability measure Q. More specifically, we have that

$$\Pi(0;X) = \frac{1}{1+R}\mathbb{E}^Q[X]\,.$$

Example 7.2. Consider a stock currently with price $S_0 = 50$. The Stock's price is expected to increase to 60 or decrease to 40 during the next year. The risk-free interest rate is 5% compounded annually. Compute the current price of a 1-year European call option with a strike price of 55.

Solution.

```
S0 <- 50
Sd <- 40
Su <- 60
R <- 0.05
K <- 55
u <- Su/S0
d <- Sd/S0

# Note that d <= (1 + R) <= u is satisfied
d
```

```
## [1] 0.8
```

```
1 + R
```

```
## [1] 1.05
```

```
u
```

```
## [1] 1.2
```

```
# Risk-neutral probabilities
qu <- (1 + R - d)/(u - d)
qu
```

```
## [1] 0.625
```

```
qd <- (u - (1 + R))/(u - d)
qd
```

```
## [1] 0.375
```

```
# or
qd <- 1 - qu
qd
```

```
## [1] 0.375
```

```
phiu <- max(Su - K, 0)
phiu
```

```
## [1] 5
```

```
phid <- max(Sd - K, 0)
phid
```

```
## [1] 0
```

```
# Current price, pi(0;X):
price <- (phiu * qu + phid * qd)/(1 + R)
price
```

```
## [1] 2.976
```

7.2.2 Multiperiod binomial model

We now extend the one-period binomial model to multiperiod. To do so, we now let the time index t run from $t = 0$ to $t = T$, where T is fixed. As previously, we have a bond with price B_t and a stock with price S_t at time t. More specifically, the bond prices are deterministic and given by

$$\begin{aligned} B_0 &= 1\,, \\ B_{n+1} &= (1+R)B_n\,, \quad n = 0, \ldots, T-1 \end{aligned}$$

On the other hand, if at time $t = 0$ the price of the stock is S_0, then the future stock prices are random and given by

$$S_{n+1} = Z_n S_n\,, \quad n = 0, \ldots, T-1$$

where $Z_0, ..., Z_{T-1}$ are iid random variables, taking only the two values u and d with probabilities

$$\mathbb{P}(Z_n = u) = p_u\,, \quad \mathbb{P}(Z_n = d) = p_d\,.$$

In other words, during each time step, the stock price either moves up to u times its initial value or moves down to d times its initial value. Figure 7.2 shows the behavior of the stock price for $T = 2$.

Note that the prices of the stock at time t can be written as

$$S_t = u^k d^{t-k} S_0\,, \quad k = 0, \ldots, t\,,$$

where k denotes the number of up-moves that have occurred. Thus, each node in the binomial tree can be represented by a pair (t, k) with $k = 0, \ldots, t$.

Figure 7.3 shows the representation of the nodes of a binomial tree for $T = 2$.

Recall that the main aim is to find the arbitrage-free price of a given contingent claim (a financial derivative, for instance). In particular, we will only work with contingent claims of the form

$$X = \Phi(S_T)\,,$$

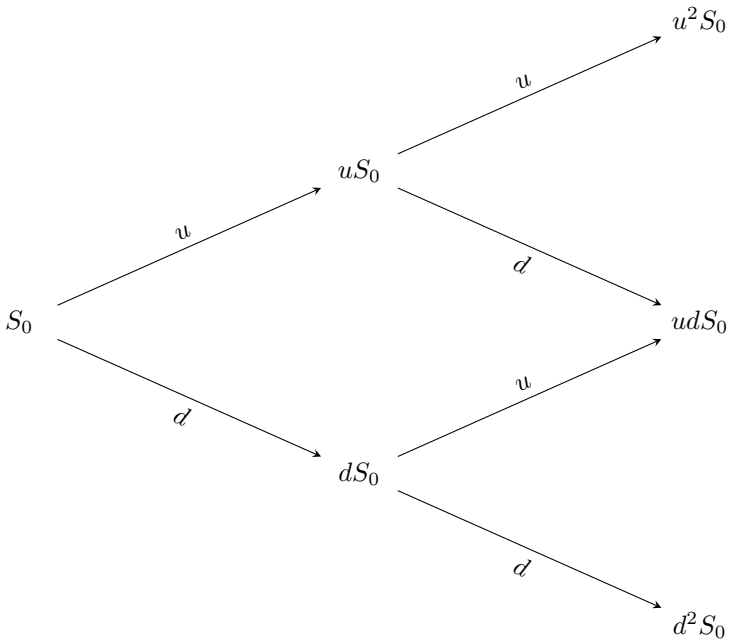


Figure 7.2: Multiperiod binomial tree.

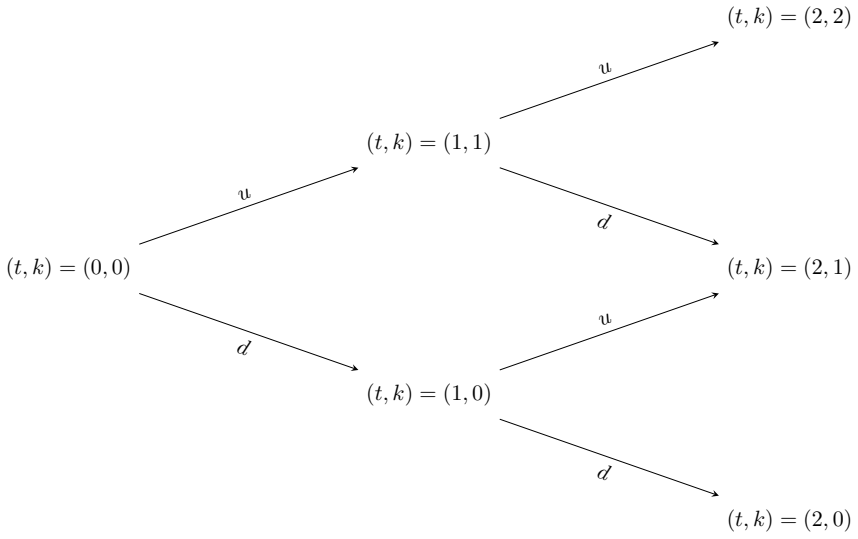


Figure 7.3: Nodes of a multiperiod binomial tree.

that is, **claims whose value only depends on the stock price at the final time** T, S_T. This type of contingent claims is typically called *simple*. Note that the European Call and Put options are examples of simple claims.

As in the one-period model, the problem is solved by finding a portfolio h that replicates the final value of the contingent claim, that is, if V_t^h denotes the value of the portfolio at time t, we require that (recall the definition of contingent claim)

$$V_T^h = X = \Phi(S_T)\,.$$

Then, the price of the derivative $\Pi(0;X)$ at time $t=0$ must be the price of the portfolio V_0^h to avoid arbitrage opportunities. More specifically,

$$\Pi(0;X) = V_0^h\,.$$

It turns out that to find the price of this portfolio (and hence of the claim), we only need to repeatedly and recursively apply the principles of the one-step binomial model. We illustrate this in a binomial model 2 periods, i.e., $T=2$. First, we need to introduce further notation. We denote by $V_t(k)$ the value of the replicating portfolio of the claim at the node (t,k). Next, we need to split the tree in Figure 7.3 into two (one-period) sub trees as shown in Figures 7.4 and 7.5.

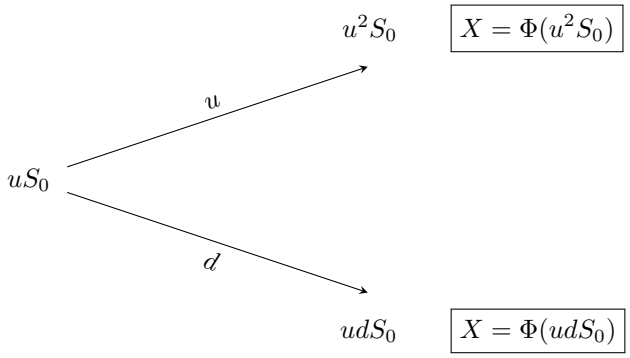


Figure 7.4: One-period subtree (up).

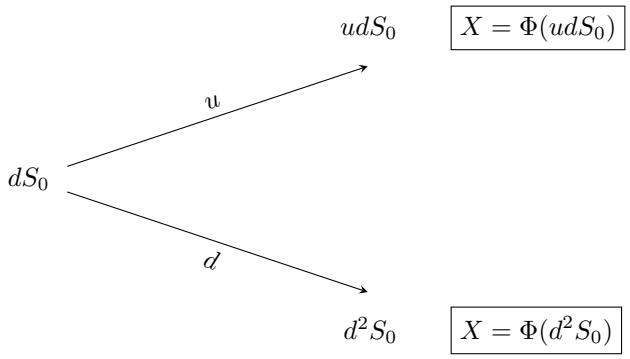


Figure 7.5: One-period subtree (down).

Remark. Note that u^2S_0, udS_0 and d^2S_0 are the values of S_t and $X = \Phi(S_t)$. Thus, one have $X = \Phi(u^2S_0), X = \Phi(udS_0)$ and $X = \Phi(d^2S_0)$ (confirm with Figures 7.4 and 7.5).

Then, we can apply the principles of the one-period binomial model to compute $V_1(1)$ and $V_1(0)$ as follows

$$V_1(1) = \frac{1}{(1+R)}(q_u\Phi(u^2S_0) + q_d\Phi(udS_0))\,,$$
$$V_1(0) = \frac{1}{(1+R)}(q_u\Phi(udS_0) + q_d\Phi(d^2S_0))\,,$$

where q_u and q_d are the risk neutral probabilities given by

$$q_u = \frac{(1+R)-d}{u-d}\,,$$
$$q_d = \frac{u-(1+R)}{u-d}\,.$$

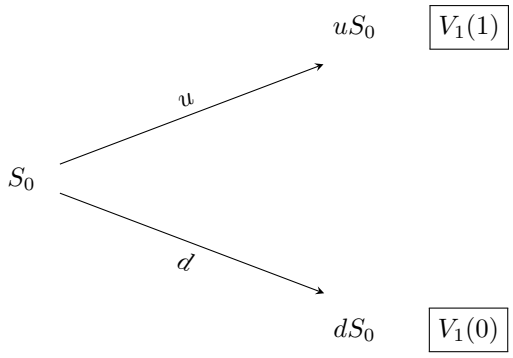


Figure 7.6: One-period subtree.

Thus, we now obtain the one-period binomial tree given in Figure 7.6.

Please notice that $V_1(1)$ is computed using u^2S_0 and udS_0. Also, $V_1(0)$ is computed using d^2S_0 and udS_0. So, it seems natural to compute $V_0(0)$ using the information on $V_1(0)$ and $V_1(1)$. Finally, using this, we compute the initial price of the portfolio $V_0(0)$ as

$$V_0(0) = \frac{1}{(1+R)}(q_u V_1(1) + q_d V_1(0)).$$

Example 7.3. Consider a stock currently with price $S_0 = 50$. The Stock's price is expected to increase to 10% or decrease 8% during the next two six-month periods. The risk-free interest rate is 5% compounded annually. Compute the price of a 1-year European put option with a strike price of 55.

Solution.

```
S0 <- 50
u <- 1.1
d <- 0.92
K <- 55

# Risk-free rate for the period of six months = (1 + r/n)^p - 1, where: r =
# risk-free interest rate compounded annually n = compounding periods in a year
# (2 semesters = 1 year); n = 2 p = number of compounding periods the rate is
# required for (1 semester); p = 1

R <- (1 + 0.05/2)^1 - 1
R
```

```
## [1] 0.025
```

```
# Number of periods (2 semesters)
n <- 2

# Future stock prices: S_t=u^k.d^(t-k).S0
k <- 0:n
St <- u^k * d^(n - k) * S0
St
```

```
## [1] 42.32 50.60 60.50
```

```
# Risk-neutral probabilities
qu <- (1 + R - d)/(u - d)
qu
```

```
## [1] 0.5833
```

```
qd <- (u - (1 + R))/(u - d)
qd
```

```
## [1] 0.4167
```

```
# or
qd <- 1 - qu

# Value of the contingent claim at maturity: X = phi(S_T); T = 2 pmax: parallel
# maxima between two vectors Eg: X1=(2,5,7) and X2=(4,0,1).
# pmax(X1,X2)=(4,5,7).
phi <- pmax(K - St, 0)
phi
```

```
## [1] 12.68  4.40  0.00
```

```
# Note: phi[1] -> k=0 (k is the number of up-moves) phi[2] -> k=1 phi[3] -> k=2


# Price of the option

# V_1(0)
vu <- (qu * phi[3] + qd * phi[2])/(1 + R)
vu
```

```
## [1] 1.789
```

```
# V_1(1)
vd <- (qu * phi[2] + qd * phi[1])/(1 + R)
vd
```

```
## [1] 7.659
```

```
# V_0(0)
v0 <- (qu * vu + qd * vd)/(1 + R)
v0
```

```
## [1] 4.131
```

We now give the general binomial algorithm.

Proposition 7.1. *Consider a claim $X = \Phi(S_T)$. Then $V_t(k)$ can be computed recursively as*

$$V_t(k) = \frac{1}{(1+R)}\left(q_u V_{t+1}(k+1) + q_d V_{t+1}(k)\right),$$
$$V_T(k) = \Phi(u^k d^{T-k} S_0).$$

In other words, the binomial algorithm consists of the following steps:

1. Generate the tree of stock prices.
2. Calculate the value of the claim at each final node.
3. Calculate the claim values sequentially at each preceding node.

To generate the tree of stock prices (step 1.), we can employ the following code:

```
build_tree <- function(S0, u, d, n) {
    tree <- matrix(0, nrow = n + 1, ncol = n + 1)
    for (t in 1:(n + 1)) {
        k <- 0:(t - 1)
        tree[t, 1:t] <- u^k * d^(t - 1 - k) * S0
    }
    tree
}
```

If we consider the same input data as in Example 7.3, we obtain the following tree

```
tree <- build_tree(50, 1.1, 0.92, 2)
tree
```

```
##       [,1] [,2] [,3]
## [1,] 50.00  0.0  0.0
## [2,] 46.00 55.0  0.0
## [3,] 42.32 50.6 60.5
```

```
# Read the matrix 'by rows' and draw the tree.
```

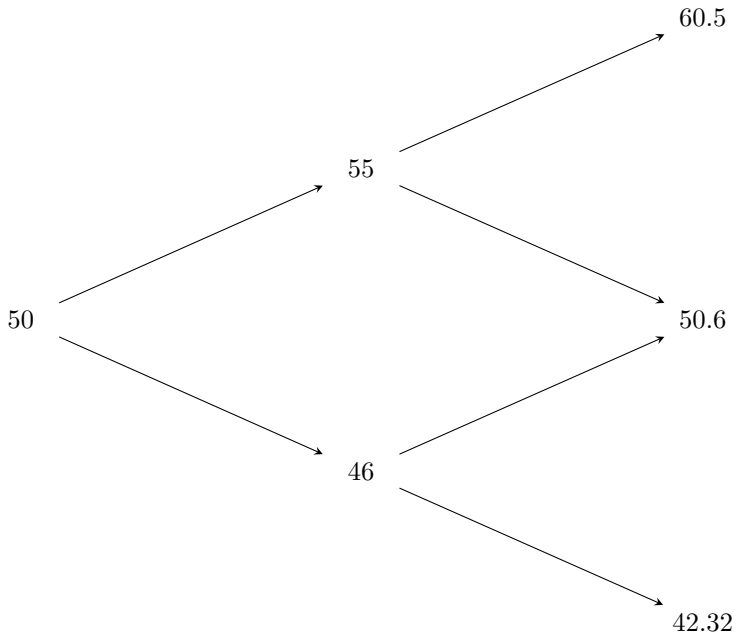


Figure 7.7: Tree for the Example 3.5.

With the above tree at hand, we can evaluate the claim at the final nodes (step 2.) by simply taking the last row of the matrix. For instance,

```
# X=Phi(S_T)=max(K-S_T,0)
pmax(K - tree[nrow(tree), ], 0)
```

```
## [1] 12.68  4.40  0.00
```

Finally, for the recursive computations (step 3.) we can use the following code:

```
value_bin_mod <- function(qu, R, tree, K) {
    val_tree <- matrix(0, nrow = nrow(tree), ncol = ncol(tree))
    val_tree[nrow(tree), ] <- pmax(K - tree[nrow(tree), ], 0)  # European put
    for (t in (nrow(tree) - 1):1) {
        for (k in 1:t) {
            val_tree[t, k] <- ((1 - qu) * val_tree[t + 1, k] + qu * val_tree[t +
                1, k + 1])/(1 + R)
        }
    }
    val_tree
}
```

Applying the above code to our example we obtain:

```
opt_price <- value_bin_mod(qu, R, tree, K)
opt_price
```

```
##         [,1]  [,2] [,3]
## [1,]  4.131 0.000    0
## [2,]  7.659 1.789    0
## [3,] 12.680 4.400    0
```

Note that the price at time $t = 0$ is given by the entry `[1, 1]`, that is,

```
opt_price[1, 1]
```

```
## [1] 4.131
```

Remark. The above code for performing the binomial algorithm was presented using a European put option as an example. However, it can be easily modified to price any other simple derivative.

It turns out that Proposition 7.1, implies the following risk-neutral valuation formula.

Proposition 7.2. *The arbitrage-free price at $t = 0$ of a claim X with maturity T is given by*

$$\Pi(0;X) = \frac{1}{(1+R)^T} \sum_{k=0}^{T} \binom{T}{k} q_u^k q_d^{T-k} \Phi(S_0 u^k d^{T-k}) \,.$$

Thus, an alternative solution to Example 7.3 using the above result is the following:

```
pi0 <- sum(choose(n, k) * qu^k * (1 - qu)^(n - k) * phi)/(1 + R)^n
pi0
```

```
## [1] 4.131
```

Remark. Sometimes, instead of u and d a volatility of the stock σ is provided. This stock's volatility is defined as the annualized standard deviation of the stock return. Luckily, given the volatility σ, we can calculate u and d by using

$$u = \exp(\sigma/n) \quad \text{and} \quad d = \exp(-\sigma/n) = 1/u \,,$$

where n is the number of intervals over one year. One can show that when n tends to infinity, the binomial model converges to the Black-Scholes model (to see ahead).

R packages for the binomial model

There are some R packages that have implementations for the binomial model. However, only important examples such as call and put options are available. Thus, when dealing with less standard derivatives, the previous analysis is highly relevant. We will come back to more complex examples shortly, but first let us illustrate the use of the `derivmkts` package (go to RStudio and download package vignettes).

```
library(derivmkts)
```

To compute prices of European (and American) call and put options we make use of the function `binomopt()`. Let us solve Example 7.3 using this function. We need to note the following about `binomopt()`: the risk-free rate (`r`) must be an annual **continuously** compounded rate ($r = r_{continuous} = log(1+R)$). To price an European option we need to change the default value of the argument `american = TRUE` to `FALSE`. By default, the function works with a volatility, hence we need to use `specifyupdn = TRUE` to indicate that we will provide u and d through the arguments `up` and `dn`. We describe the rest of the arguments needed in the following code:

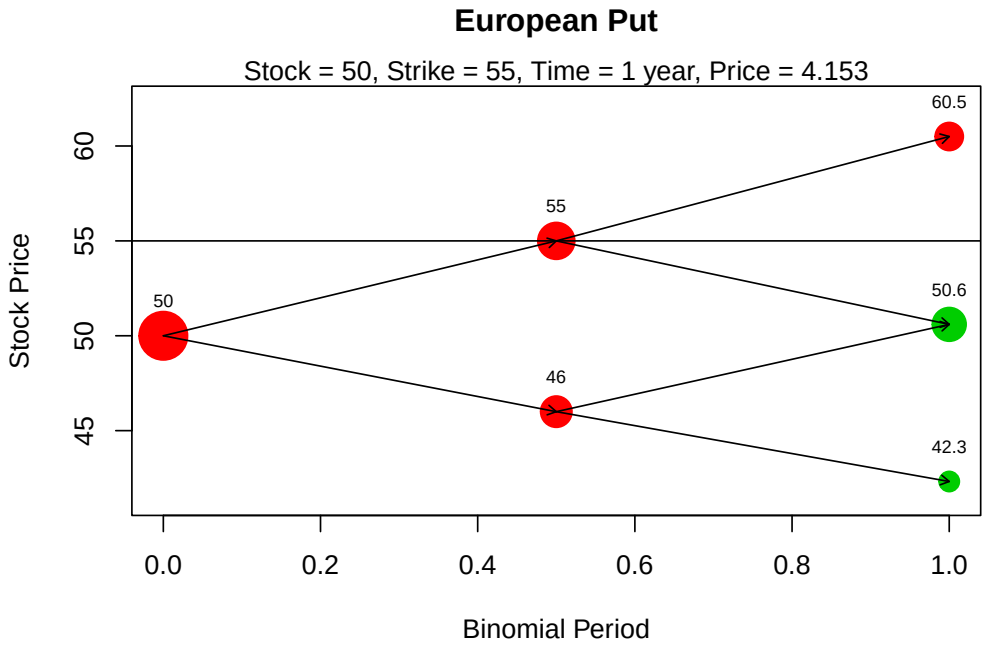
```
binomopt(
  s = 50,
  # Initial stock price
  k = 55,
  # Strike price
  r = log(1 + 0.05),
  # Continuously-compounded risk-free rate
  tt = 1,
  # Time to maturity
  d = 0,
  # Dividends, in our case, we do not work with dividends, hence 0.
  nstep = 2,
  # Number of periods
  american = FALSE,
  # To indicate European
  putopt = TRUE,
  # To indicate a Put option
  specifyupdn = TRUE,
  # Tells the function to use u and d
  up = 1.1,
  # Value of u
  dn = 0.92
  # Value of d
)
```

```
## price
## 4.153
```

Remark. The value 4.153 differs from 4.131 due to the use of a continuously-compounded risk-free rate.

Moreover, `derivmkts` contains the function `binomplot()` which plots the development of the stock price and shows graphically the probability of being at each node (represented as the area of the circle at that price). The following is the plot for our example:

```
binomplot(
  s = 50, k = 55, r = log(1 + 0.05), tt = 1, d = 0, nstep = 2,
  american = FALSE, putopt = TRUE, specifyupdn = TRUE, up = 1.1, dn = 0.92,
  v = 0, # A value of the volatility must be provided, although not used
  plotarrows = TRUE, # Plots arrows that connect the nodes of the tree
  plotvalues = TRUE # Plots the values of the stock prices at each node
)
```



Note that a horizontal line with the strike value is included in the plot. This can also be omitted by using `drawstrike = FALSE`. Finally, the green and red colors indicate whether or not the option is optimally exercised there (green if yes, red if no).

We finish our discussion of the binomial model with the following more complicated example, which cannot be solved directly with the functions in `derivmkts`. Hence the importance of our early discussion.

Example 7.4. A stock price is currently 40. Over each of the next three 5-month periods, it is expected to go up by 10% or down by 5%. The risk-free interest rate is 6% per annum with continuous compounding. Apply Proposition 7.1 and a three-step binomial model to find the value of a European style derivative that pays off $X = [\max(K - S_T, 0)]^3$, where S_T is the stock price in 15 months and $K = 44$.

Solution.

```
s0 <- 40
u <- 1.1
d <- 0.95
n <- 3
K <- 44
tree <- build_tree(s0, u, d, n)
tree
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 40.00  0.00  0.00  0.00
## [2,] 38.00 44.00  0.00  0.00
## [3,] 36.10 41.80 48.40  0.00
## [4,] 34.29 39.71 45.98 53.24
```

```
pmax(K - tree[nrow(tree), ], 0)
```

```
## [1] 9.705 4.290 0.000 0.000
```

```
R <- exp(0.06 * 5/12) - 1
# Rcont=log(1+R) <=> R=exp(0.06)-1 (annually) => R=exp(0.06*5/12)-1 for a
# 5-month period.
R
```

```
## [1] 0.02532
```

```
qu <- (1 + R - d)/(u - d)
qu
```

```
## [1] 0.5021
```

```
# We cannot use the original function value_bin_mod or package derivmkts!!
# X=phi(S_T) is now [max(K-S_T;0)]^3

value_bin_mod.new <- function(qu, R, tree, K) {
    val_tree <- matrix(0, nrow = nrow(tree), ncol = ncol(tree))
    val_tree[nrow(tree), ] <- (pmax(K - tree[nrow(tree), ], 0))^3  # given claim
    for (t in (nrow(tree) - 1):1) {
        for (k in 1:t) {
            val_tree[t, k] <- ((1 - qu) * val_tree[t + 1, k] + qu * val_tree[t +
                1, k + 1])/(1 + R)
        }
    }
    val_tree
}


opt_price <- value_bin_mod.new(qu, R, tree, K)
opt_price
```

```
##       [,1]  [,2] [,3] [,4]
## [1,] 132.0  0.00    0    0
## [2,] 253.1 18.62    0    0
## [3,] 482.5 38.34    0    0
## [4,] 914.1 78.95    0    0
```

Thus, the price of the option is

```
opt_price[1, 1]
```

```
## [1] 132
```

7.3 The Black-Scholes model

The binomial model discussed in the previous section is a discrete-time model: the stock price changes at the end of each time period. Another commonly used model for option pricing is the Black-Scholes model, which assumes that the stock price moves continuously on time. The assumptions behind the Black-Scholes model are deep, and in fact, an entire course can be dedicated to the development of this model. In this course, it is sufficient for you to know how to price the options using simulations and the closed-form Black-Scholes formula. We start with a review of the Brownian motion, which will be used to describe the price movements of a stock.

7.3.1 Preliminars: Brownian motion

Definition 7.4 (Stochastic process)**.** A *stochastic process* is a family of random variables $\{X(t),\ t \in T\}$ defined on a probability space $(\Omega, \mathcal{F}, P)$, where:

- t represents, typically, the time.
- T is the set where the time parameter t is defined.
- Ω is the set of all possible outcomes.
- $\mathcal{F}$ is a set of events.
- P is a probability function.
- $(\Omega, \mathcal{F}, P)$ will become clear in "Probability Theory and Stochastic Processes".

If $T = \mathbb{N}$, we say that the process is a discrite time process; if $T = [a,\ b] \subseteq \mathbb{R}$, we say it is a continuous time process.

Remark.

- $\{X(t),\ t \in T\} = \{X(t,\omega),\ t \in T,\ \omega \in \Omega\}$.
- $X(t) = X_t$ is called the sate or position of the process at time t.
- The space of the states, denoted by $\mathcal{S}$, is the space where the random variables take values.
- If $\mathcal{S} = \mathbb{R}$ the space is called continuous state space.
- If $\mathcal{S} = \mathbb{N}$ the space is called discrete state space.
- For each fixed $\omega \in \Omega$, the mapping $t \to X(t,\omega)$ is called a realization, trajectory or sample path of the process.

Definition 7.5 (Standard Brownian motion)**.** A *standard Brownian motion* (SBM), also known as standard Wiener process, is a stochastic process $W = (W(t))_{t\geq 0}$ satisfying:

1. $W(0) = 0$.
2. The process has *independent increments*, i.e., for any $0 \leq t_1 < t_2 < \cdots < t_n$,
$$W(t_2) - W(t_1), \ldots, W(t_n) - W(t_{n-1})$$
are independent random variables.
3. For $s < t$, $Y = W(t) - W(s) \sim N(\mu_Y = 0, \sigma_Y^2 = t - s)$.
4. W has continuous trajectories.
5. $Cov(W_t, W_s) = min(t, s)$.

Remark. With $s = 0$, it follows from 3. that $W(t) \sim N(0, t)$. Notice that the variance is proportional to the time interval.

To generate trajectories of a SBM over a time period $[0, T]$, we consider a "small" time increment $\delta t > 0$. Then, consider the independent increments

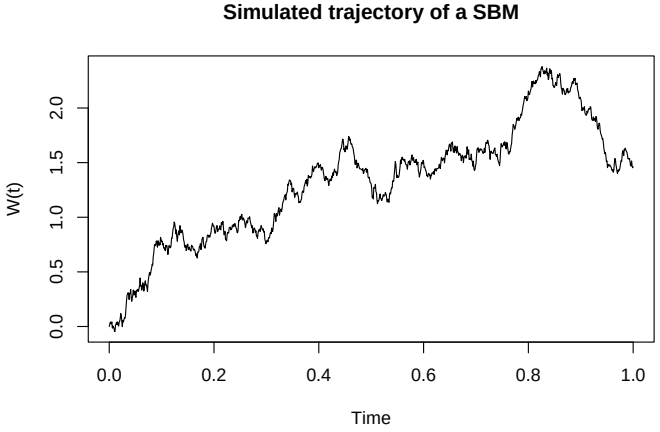
$$W(\delta t) - W(0)\,,$$
$$W(2\delta t) - W(\delta t)\,,$$
$$\ldots\,.$$

Note that these increments are $N(0, \delta t)$ distributed. Moreover, we can compute $W(2\delta t)$ as

$$W(2\delta t) = (W(2\delta t) - W(\delta t)) + (W(\delta t) - W(0))\,.$$

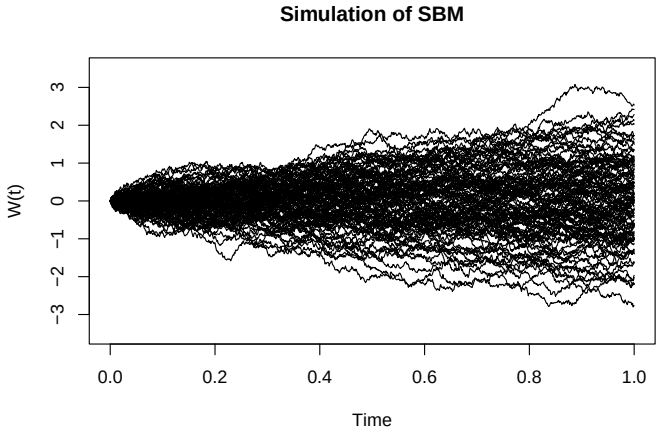
Thus, we can repeat the same logic as above to simulate a whole process trajectory. The following code implements this idea:

```
set.seed(12345)
delta <- 0.001  # Time increment
t <- seq(0, 1, by = delta)  # Time interval
# rnorm(n, mean = 0, sd = 1)
w <- rnorm(n = length(t) - 1, mean = 0, sd = sqrt(delta))  # iid normal distributed r.v.s
w <- c(0, cumsum(w))  # Cumulative sum; 0 is the initial value
plot(t, w, type = "l", xlab = "Time", ylab = "W(t)", main = "Simulated trajectory of a SBM",
    )
```



We now modify the code above to generate multiple trajectories:

```
nsim <- 100 # Number of simulated trajectories
w <- matrix(
  rnorm(n = nsim*(length(t)-1), mean = 0, sd = sqrt(delta)),
  nsim, # Each row is one simulated trajectory
  length(t) - 1
)
w <- cbind(rep(0, nsim), t(apply(w, 1, cumsum))) # matrix with trajectories
plot(t, w[1, ], # Plots the first trajectory
  type = "l",
  ylim = c(-3.5, 3.5),
  xlab = "Time",
  ylab = "W(t)",
  main = "Simulation of SBM"
)
apply(w[-1, ], 1, function(x, t) lines(t, x), t = t) # Plots the remaining trajectories
```



The standard Brownian motion can be generalized to the *arithmetic Brownian motion*, which scales and shifts the former. More specifically, $X(t)$ is an arithmetic Brownian motion if

$$X(t) = \mu t + \sigma W(t),$$

where $W(t)$ is a standard Brownian motion, $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$. Moreover, note that for $s < t$,

$$X(t) - X(s) \sim N(\mu(t-s), \sigma^2(t-s)).$$

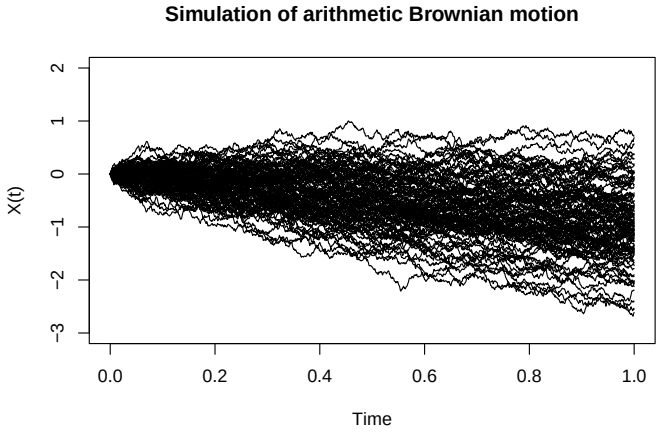
To simulate from a standard Brownian motion an easier way is to modify in our code above and generate independent $N(\mu\delta t, \sigma^2\delta t)$ random variables. The following code implements the simulation:

- first we simulate data:

```
mu <- -1
sigma2 <- 0.5
nsim <- 100
x <- matrix(rnorm(n = nsim * (length(t) - 1), mean = mu * delta, sd = sqrt(delta *
    sigma2)), nsim, length(t) - 1)
x <- cbind(rep(0, nsim), t(apply(x, 1, cumsum)))
```

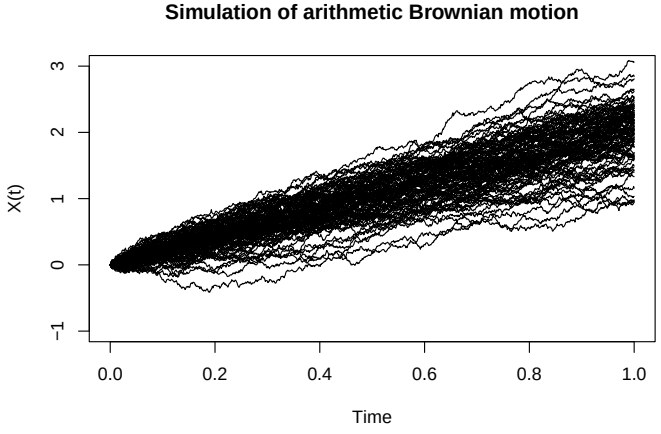
- secondly, we plot data:

```
plot(t, x[1, ],
  type = "l",
  ylim = c(-3, 2),
  xlab = "Time",
  ylab = "X(t)",
  main = "Simulation of arithmetic Brownian motion"
)
apply(x[-1, ], 1, function(x, t) lines(t, x), t = t)
```



Experiment with different μ and σ^2 values:

```
mu <- 2
sigma2 <- 0.2
nsim <- 100
x <- matrix(
  rnorm(n = nsim * (length(t) - 1), mean = mu * delta, sd = sqrt(delta * sigma2)),
  nsim,
  length(t) - 1
)
x <- cbind(rep(0, nsim), t(apply(x, 1, cumsum)))
plot(t, x[1, ],
  type = "l",
  ylim = c(-1, 3),
  xlab = "Time",
  ylab = "X(t)",
  main = "Simulation of arithmetic Brownian motion"
)
apply(x[-1, ], 1, function(x, t) lines(t, x), t = t)
```



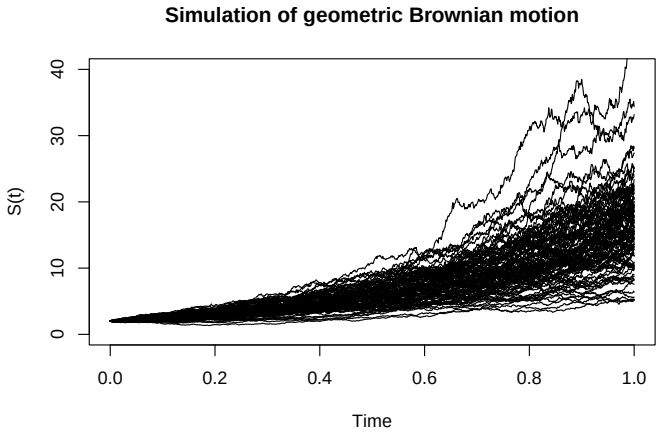
Note that the arithmetic Brownian motion can take negative values. Hence, using it for modeling stock prices is questionable. Instead, we now introduce the so-called called *geometric Brownian motion*, which can only take non-negative values. More precisely, a geometric Brownian motion, denoted by $S(t)$, is a stochastic process of the form

$$S(t) = S(0) \exp\left(X(t)\right) = S(0) \exp\left(\mu t + \sigma W(t)\right),$$

where $S(0) > 0$ is some known starting value, and $X(t)$ is an arithmetic Brownian motion. The following code generates some trajectories of this process using the previous implementation:

```
s0 <- 2
s <- s0 * exp(x)
plot(t, s[1, ],
  type = "l",
  ylim = c(0, 40),
  xlab = "Time",
```

```
  ylab = "S(t)",
  main = "Simulation of geometric Brownian motion"
)
apply(s[-1, ], 1, function(x, t) lines(t, x), t = t)
```



Note that in particular,

$$\frac{S(t)}{S(0)} \sim LN(\mu t, \sigma^2 t),$$

and more generally, for $s < t$,

$$\frac{S(t)}{S(s)} \sim LN(\mu(t-s), \sigma^2(t-s)).$$

Remember: $Y \sim LN(.,.) \iff \log(Y) \sim N(.,.)$.

Finally, one can show (to see next year!) that a geometric Brownian motion of the form

$$S(t) = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W(t)\right),$$

satisfies the *stochastic differential equation* (SDE)

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t), \quad S(0) = S_0,$$

known as the Black-Scholes model.

7.3.2 The Black-Scholes formula

As in the binomial model, the Black–Scholes model assumes that the prices of these two assets satisfy

$$\begin{aligned} dB(t) &= rB(t)dt, \\ dS(t) &= rS(t)dt + \sigma S(t)d\overline{W}(t), \quad S(0) = S_0, \end{aligned} \tag{7.1}$$

where

- $B(t)$ represents the deterministic price of a riskless asset (a bond or a bank deposit);
- $S(t)$ is the stochastic price process of a risky asset (a stock or an index);
- $\overline{W}$ is a standard Brownian motion with respect to the original probability measure P;
- r is the risk-free interest rate;
- σ represents the volatility of the risky asset, and
- S_0 is the (known) initial condition.

The solution of (7.1) is the geometric Brownian motion:

$$S(t) = S_0 \exp\left(\left(r - \frac{\sigma^2}{2}\right)t + \sigma\overline{W}(t)\right).$$

Now, we consider a contingent claim (a financial derivative) of the form

$$X = \Phi(S(T)).$$

It can be shown that the arbitrage-free price $\Pi(0;X)$ of X is given by the risk-neutral evaluation formula

$$\Pi(0;X) = e^{-rT}\mathbb{E}^Q\left[\Phi(S(T))\right], \tag{7.2}$$

where the behavior of S under Q is described by the SDE

$$dS(t) = rS(t)dt + \sigma S(t)dW(t),$$

whose solution is the geometric Brownian motion and where $W(t)$ denotes the Brownian motion under the measure Q. In particular, we have that $S(T)$ is explicit and given by

$$S(T) = S_0 \exp\left(\left(r - \frac{\sigma^2}{2}\right)T + \sigma W(T)\right). \tag{7.3}$$

When considering a European call option, that is, $\Phi(x) = \max(x-K,0)$, (7.2) and (7.3) lead to the famous **Black–Scholes formula** as follows:

Proposition 7.3 (Black-Scholes formula)**.** *The price of an European call option $C(0)$ with strike price K and time of maturity T is given by*

$$C(0) = S_0 N(d_1) - Ke^{-rT}N(d_2), \tag{7.4}$$

where N is the cumulative distribution function for the $N(0,1)$ distribution and

$$d_1 = \frac{1}{\sigma\sqrt{T}}\left(\log\left(\frac{S_0}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right)T\right),$$
$$d_2 = d_1 - \sigma\sqrt{T}.$$

Similarly, the price of a European put option $P(0)$ with strike price K and time of maturity T is given by

$$P(0) = Ke^{-rT}N(-d_2) - S(0)N(-d_1),$$

with d_1 and d_2 as above.

$N(d_1)$ and $N(d_2)$ are obtained in R as `pnorm(d1)` and `pnorm(d2)`.

Remark. Using Itô's formula (next year, stochastic calculus), one can get an expression for the BS formula as a PDE (already seen in MMF – Part I).

In R, the Black–Scholes formula to price European call and put option is available in the `derivmkts` package under the functions `bscall()` and `bsput()`, respectively. We illustrate its use with an example.

Example 7.5. Consider a European call option over a stock with a current price S(0) = 50 and volatility 0.25. Moreover, the risk-free interest rate with continuous compounding is 6% per annum, the strike price is 45, and the option's time to maturity is 6 months. Find the price of the option.

Solution.

```
# bscall(s, k, v, r, tt, d) s: Price of the underlying asset k: Strike price v:
# Volatility of the asset price, defined as the annualized standard deviation
# of the continuously-compounded return r: Annual continuously-compounded
# risk-free interest rate tt: Time to maturity in years d: Dividend yield,
# annualized, continuously-compounded

s <- 50
k <- 45
v <- 0.25
r <- 0.06
tt <- 6/12
d <- 0  # No dividends

bscall(s, k, v, r, tt, d)
```

```
## [1] 7.382
```

7.3.3 Greeks

Greeks represent sensitivities of a derivative value to changes in the underlying parameters used to determine its price. More specifically, if we denote by V be the price of a derivative, which depends on the underlying stock price S, a risk-free rate r, a volatility σ and with maturity T, then the Greeks are defined as follows:

Definition 7.6 (Greeks)**.**

Delta, Δ: Measures the rate of change of the option price with respect to changes in the underlying stock price. [similar to velocity]

$$\Delta = \frac{\partial V}{\partial S}\,.$$

Gamma, Γ: Measures the rate of change in Δ with respect to changes in the underlying stock price. [similar to acceleration]

$$\Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{\partial \Delta}{\partial S}\,.$$

Vega, ν: Measures sensitivity with respect to changes in the volatility σ.

$$\nu = \frac{\partial V}{\partial \sigma}\,.$$

Rho, ρ: Measures sensitivity with respect to changes in the risk-free interest rate.

$$\rho = \frac{\partial V}{\partial r}\,.$$

Theta, Θ: Measures the sensitivity of the value of the derivative to the passage of time. [Theta is generally expressed as a negative number and can be thought of as the amount by which an option's value declines every day.]

$$\Theta = -\frac{\partial V}{\partial T}\,.$$

Elasticity or Lambda, λ: Is the percentage change in option value per percentage change in the underlying price (a measure of leverage).

$$\lambda = \frac{\partial V}{\partial S}\frac{S}{V} = \Delta \times \frac{S}{V}\,.$$

Psi, ψ: Is the percentage change in option value per percentage change in the underlying dividend yield (q).

$$\psi = \frac{\partial V}{\partial q}\,.$$

In particular, for a European call option with strike price K and time of maturity T we have the following explicit expressions:

$$\begin{aligned}
\Delta &= N(d_1)\,,\\
\Gamma &= \frac{N'(d_1)}{S\sigma\sqrt{T}}\,,\\
\nu &= S\sqrt{T}N'(d_1)\,,\\
\rho &= KTe^{-rT}N(d_2)\,,\\
\Theta &= -\frac{S(0)N'(d_1)\sigma}{2\sqrt{T}} - rKe^{-rT}N(d_2)\,.
\end{aligned}$$

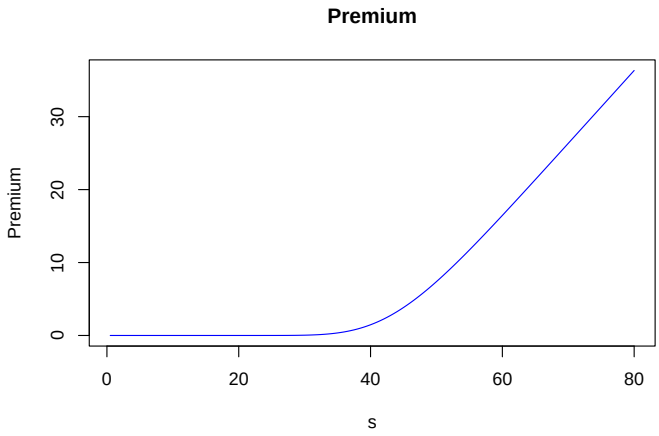
The `derivmkts` R package allows us to compute the Greeks for a financial derivative via the function `greeks()`. For instance, the Greeks for the option in Example 7.5 can be computed as follows:

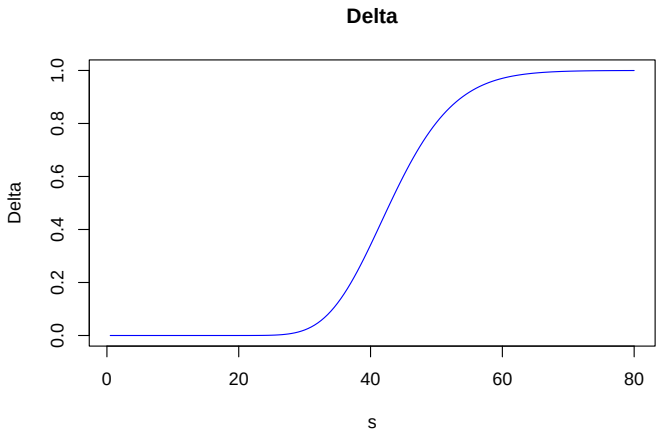
```
greeks(bscall(s, k, v, r, tt, d = 0))
```

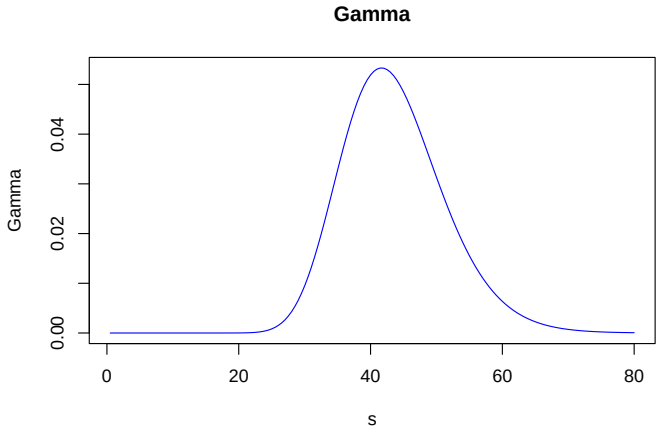
```
##              bscall
## Premium     7.38194
## Delta       0.80348
## Gamma       0.03134
## Vega        0.09794
## Rho         0.16396
## Theta      -0.01210
## Psi        -0.20087
## Elasticity  5.44217
```

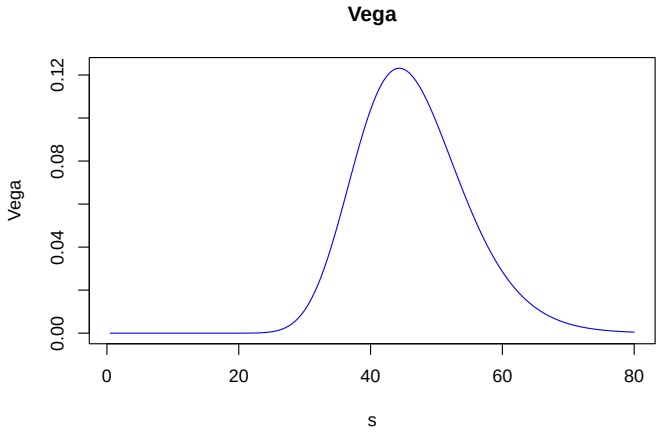
Furthermore, `greeks()` accepts vector inputs, which in particular allow us to visualize the Greeks.

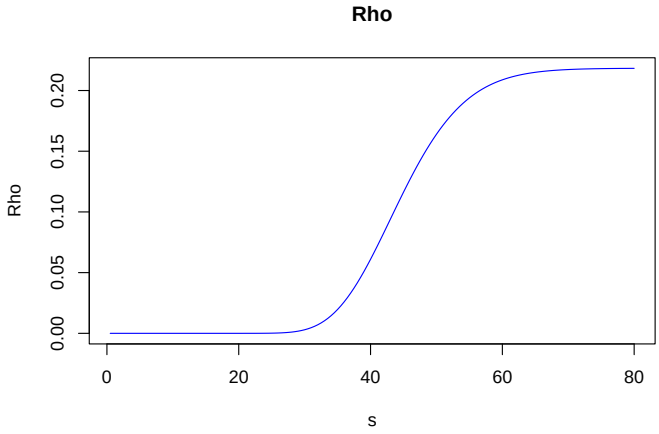
```
s <- seq(0.5, 80, by = 0.5)
call_greeks <- greeks(bscall(s, k, v, r, tt, d = 0))
for (i in rownames(call_greeks)) {
    plot(s, call_greeks[i, ], main = paste(i), ylab = i, type = "l", col = "blue")
}
```

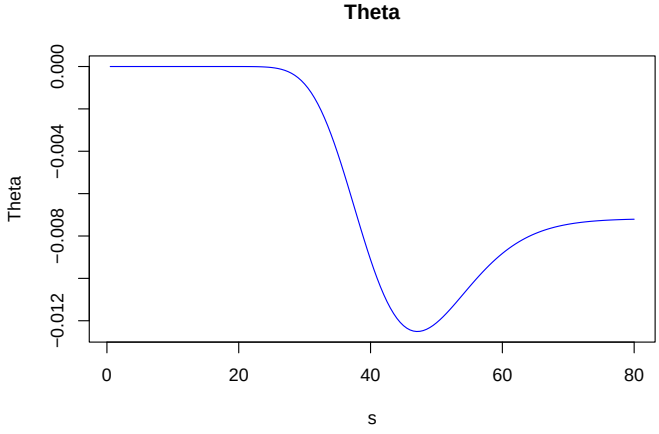


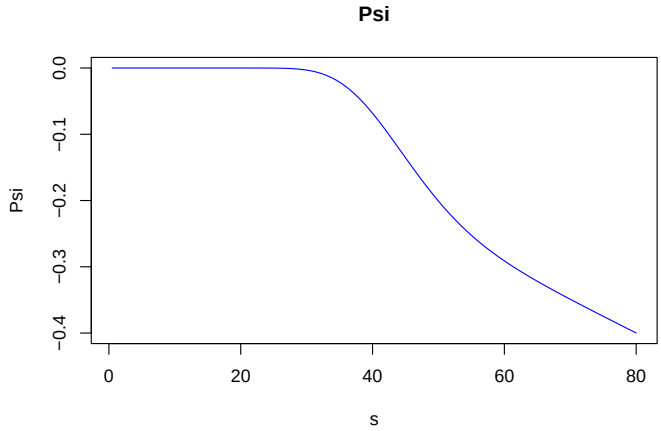


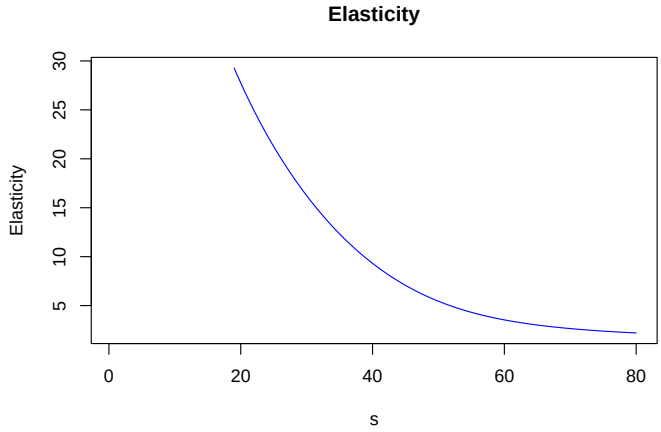

Gamma
Gamma
0.00
0.02
0.04
0
20
40
60
80
s


Vega
Vega
0.00
0.04
0.08
0.12
0
20
40
60
80
s


Rho
Rho
0.00
0.05
0.10
0.15
0.20
0
20
40
60
80
s


Theta
Theta
−0.012
−0.008
−0.004
0.000
0
20
40
60
80
s


Psi
Psi
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
s


Elasticity
Elasticity
30
25
20
15
10
5
0
20
40
60
80
s

Chapter 8

Exercises

8.1 Related to Chapters 1 to 6

1. Write an R code to determine the result of the following computation:

$$e^{\pi}\left(\log_2(10)+\frac{5^2+\sin(8)}{\sqrt{2}}\right)^{-3}.$$

2. Without using R, determine the result of the following logical computation

```
((!(4 == 3) | (abs(-3) <= 2)) & ((2^2 > 4) & (TRUE))) | ((!FALSE | (4+2) == 5) & (0.5 >= (1/2)))
```

Verify your result by typing the code in R.

3. Find the errors in the following lines of code:

a)

```
2 +     3 *4 + sqrt[100]
```

b)

```
(2 + i)  / 3  + {1e1 + 4.0i}
```

c)

```
time.To.Maturity <- 6
Interest.rate <- 0.05
{1e-0 + interest.rate}^{-time.To.Maturity}
```

4. Without using R, determine the result of the following computation:

```
x <- c(1, 2, 3)
x[2] / x[2]^2 - 1 + 3 * x[3] - x[2 - 1]
```

Verify your result by typing the code in R.

5. Write an R code to calculate the amount of money owed after n years, where n varies from 1 to 10 in yearly increments, assuming that the money lent originally is 2350 and the interest rate remains constant throughout the period at 5%.

6. Consider the vector `1:N`, where `N` is a positive integer. Write an R code that determines how many elements in the vector are exactly divisible by 4. Test your code with `N <- 40`.

7. Compute the mean of:

(a) $v = (0, 1, NA, 2, 7)$.
(b) $u = (-Inf, 0, 1, NA, 2, 7, Inf)$.

8. Create:

(a) a vector with values $e^x \cos(x)$ at $x = 3, 3.1, 3.2, \ldots, 6$.

(b) a vector with values $\left(2, \frac{2^2}{2}, \frac{2^3}{3}, \ldots, \frac{2^{25}}{25}\right)$.

9. Compute:

(a)

$$\sum_{j=10}^{100} j^3 + 4j^2.$$

(b)

$$\sum_{j=1}^{25} \frac{2^j}{j} + \frac{3^j}{j^2}.$$

10. Create the vectors:

(a) $(1, 2, 3, \ldots, 19, 20)$.
(b) $(20, 19, \ldots, 2, 1)$.
(c) $(1, 2, 3, \ldots, 19, 20, 19, 18, \ldots, 2, 1)$.
(d) $(1, 2, 3, 1, 2, 3, \ldots, 1, 2, 3)$ where there are 10 occurrences of 1.
(e) $(1, 1, \ldots, 1, 2, 2, \ldots, 2, 3, 3, \ldots, 3)$ where there are 10 occurrences of 1, 20 occurrences of 2 and 30 occurrences of 3.
(f) $(0.1^3 0.2^1, 0.1^6 0.2^4, \ldots, 0.1^{36} 0.2^{34})$.

11. Use the function `paste` to create the following character vector of length 20:

("label 1", "label 2", . . . , "label 20")

12. Set the vector `v2` equal to the following: "A" "A" "B" "B" "C" "C" "D" "D" "E" "E" (note the letters are all uppercase).

13. Set the vector `v3` equal to the following: "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" (note the letters are all lowercase).

14. Set the vector `v4` equal to the words "dog" 10 times, "cat" 9 times, "fish" 6 times, and "fox" 1 time.

15. Run the following lines:

```
> set.seed(50)
> x <- sample(0:999, 250, replace = TRUE)
> y <- sample(0:999, 250, replace = TRUE)
```

(a) Explain the meaning of x and y.

(b) Denote by x_i and y_i the entries of the vectors x and y, respectively. Create vectors $(y_2 - x_1, \ldots, y_n - x_{n-1})$ and $\left(\frac{\sin(y_1)}{\cos(x_2)}, \ldots, \frac{\sin(y_{n-1})}{\cos(x_n)}\right)$.

(c) Compute

$$\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i + 10}.$$

(d) Denote by $\bar{x}$ the mean of the vector x and compute

$$\sum_{i=1}^{n} |x_i - \bar{x}|^{1/2}.$$

(e) Extract the values of x which correspond to the values of y which are > 500.

(f) How many values of x are above the mean of y?

(g) How many values of x are smaller than the minimum value of y?

(h) How many values of x are divisible by 2?

(i) Extract the 1st minimum value of x, the 4th minimum value of x, the 7th minimum value of x, etc.

16. Create a random sequence of length 100 taking values "low", "medium" and "high". Consult the help of `sample`.

17. Create a vector containing the following student grades: 70, 80, 55, 67, 90, 92, 83, 74, 100, 87, 49. Using logical operators and vector functions, answer the following question:

a) What is the average grade of the whole group?
b) How many students have grades less than 65?
c) What is the average grade of those students with grades between 60 and 80 (including 60 and 80)?
d) Assume that we add three new students with grades 65, 98, 54. Repeat questions a)-c) with the new vector of grades.

18. Write an R program to compute the alternating harmonic series

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n},$$

up to a finite number of summands N. Test your code with $N = 100$ and compare with $\log(2)$.

19. The function `cov()` computer the sample covariance of two vectors. Recall that for two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ the sample covariance is given by

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Write an R program to compute the sample correlation without using the `cov()` function. Test your code with the following vectors: `x <- c(1, 2, 3, 4)` and `y <- c(2, 2, 3, 5)`. Verify your result using `cov(x, y)`.

20. Create three vectors of length 3, with content and names of your choice. Next, combine the three vectors into a 3×3 matrix where each column represents one of your vectors. Finally, compute the determinant (`det()` - is your matrix invertible?) and the transpose of your matrix.

21. Consider the following matrix:

```
matrix(c(c(-4, 2, 1), c(0, -1, 0.5), c(1.5, 0.2, -2)), ncol = 3, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,] -4.0  2.0  1.0
## [2,]  0.0 -1.0  0.5
## [3,]  1.5  0.2 -2.0
```

Compute the sum of rows (that is, you have to return a vector of length three with first entry -1) via the following three methods:

a) Using the `rowSums()` function (see help for more information).
b) Using the `apply()` function.

c) Using matrix multiplication. *Hint*: This can be done by multiplying the above matrix with an appropriate vector.

22. Write an R program to:

a) Create a numeric vector called `rates` with values: 0.043, 0.045, 0.041, 0.049, 0.05, 0.055, 0.048, 0.0495, 0.051, 0.044, 0.045, 0.0455.
b) Create a character vector called `months` with values: "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec".
c) Create a data frame called `monthly_rates` using `months` and `rates`.
d) Add a new column to your data frame called `year` with values 2021 for all rows.
e) Extract the rows where the rates are above 5%.
f) Extract the rows where the rates are below the mean of the whole year.

23. Assume a group of students with ages

```
age <- c(19, 20, 18, 19, 18, 20, 18, 19, 19, 20)
```

and grades

```
grade <- c(90, 75, 80, 87, 74, 93, 100, 66, 71, 89)
```

Let us imagine that we want to compute the average grade by age. We can use the `tapply()` function to do so. Look at the documentation of `tapply()` (`?tapply`) and solve the above problem.

24. Consider the matrix

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$$

(a) Verify that $A^3 = 0$.

(b) Replace the third column by the sum of column 1 and column 2.

25. Consider the matrix

$$M = \begin{bmatrix} 33 & 51 & 60 \\ 20 & 35 & 17 \\ 15 & 27 & 24 \\ 15 & 21 & 28 \\ 14 & 25 & 29 \\ 10 & 53 & 24 \end{bmatrix},$$

and replace even numbers by -1.

26. Solve the linear system

$$\begin{cases} 2y + x - z = 1 \\ 2z - x - y = 0 \\ x + 1 + y + 2z = 1 \end{cases}$$

27. Run the following lines:

```
> set.seed(75)
> M <- matrix(sample(10, size = 60, replace = TRUE), nrow = 6)
```

(a) Explain the output.

(b) Find the number of entries in each row which are greater than 4.

(c) Which rows contain exactly two occurrences of the number 7?

(d) Denote by $M_{i,j}$ the entries of the $n \times m$ matrix M and compute the following means and sums of squares:

$$\mu = \frac{1}{n \times m} \sum_{i,j} M_{i,j}$$

$$\mu_i = \frac{1}{m} \sum_{j} M_{i,j}, \quad i = 1, \ldots, n$$

$$SS_1 = \sum_{i} (\mu - \mu_i)^2$$

$$SS_2 = \sum_{i,j} (M_{i,j} - \mu_i)^2$$

$$SS_3 = \sum_{i,j} (M_{i,j} - \mu)^2$$

28. Install and load the package `DAAG`. Execute:

(a) Type `library(help="DAAG")` to get the information on this package.

(b) Consider the dataset `carpriceand` apply the convenient `R` command to check the structure of the dataset.

(c) Set a name for `carprice`, say `x`.

(d) Explain the meaning of `table(x$Type)`.

(e) Replace all column names.

(f) Compute the mean, variance, standard deviation, range, maximum and minimum of the maximum price variable.

(g) Drop all the rows with maximum price higher than 24.8 m.u.

(h) Drop all the rows with minimum price lower than 14.0 m.u.

29. Create a list with information about 3 workers as follows:

(a) The list should have 4 vectors as components: name, age, gender and a boolean variable (it assumes values 0 or 1 that defines whether the person works or not).

(b) Create a data frame with the information from the previous list.

(c) Add a vector to the previous data frame with information about the person's job.

(d) Compute the mean, variance, standard deviation, range, maximum and minimum of the age.

30. Run the following lines:

```
> set.seed(75)
> x <- sample(10, size = 10, replace = TRUE)
> y <- sample(10, size = 10, replace = TRUE)
```

(a) Create a list whose elements are the vectors x and y with names "sample1" and "sample2", respectively.
(b) Create a list with the mean and standard deviation of each sample.

31. Create a list:

(a) with years from 2005 to 2016, 12 months and 31 days.
(b) replace year by `c(2000:2010)`.
(c) delete the value 4 of the month.

32. Write a function called `my_add` that adds two numbers (`x` and `y`) together and returns the results. Run `my_add(5,7)`.

33. Copy the function `my_add` above and add an error message that returns "x and y must be numbers" if `x` or `y` are not both numbers.

34.

(a) Write a function that accepts a numerical vector $(x_1, x_2, \ldots, x_n)$ as argument and returns the vector

$$\left(x_1, \frac{x_2^2}{2}, \ldots, \frac{x_n^n}{n}\right).$$

Run for `1:4`.

(b) Write a function that accepts a single number x and a positive integer n and returns the sum

$$1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots + \frac{x^n}{n}.$$

Run for $x = \sqrt{2}$ and $n = 5$.

(c) Write a function which takes a matrix as single argument and returns a matrix which is the same as the function argument but every odd entry is doubled. Run for `matrix(sample(1:10),2,5)`.

(d) Write a function to convert the temperature measured in Celsius (°C) to Fahrenheit (°F). Check that `f.d(0) = 32` and `f.d(-40) = -40`.

(e) Write a function that takes a single integer argument n and returns the sum of its divisors, e.g., if $n = 6$, then the function returns $1 + 2 + 3 + 6$.

(f) Write a function that takes arguments $a, b, c \in \mathbb{R}$ and returns the roots of the polynomial $ax^2 + bx + c$. The output should be a list containing the two roots. In case the roots are complex conjugated, the output of each root should be a list of its real and imaginary parts. Find the solutions of $x^2 - 4x + 4 = 0$ and $3x^2 - x + 4 = 0$.

35. Consider the function

$$f(x) = \begin{cases} 2e^{-2x}, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

(a) Show that f is a density probability function (dpf).

(b) Let X be a r.v. with dpf f and compute $P(X > 1)$. Show the value with 2 decimal places.

(c) Compute $P(0.3 < X < 0.7)$. Show the value with 3 decimal places.

(d) Compute $E[X]$. Show the value with 1 decimal place.

36. Plot a graphic of $f(x) = 1 - \frac{1}{x}sin(x)$ for $0 < x < 5$ and for $0 < x < 50$.

37. Consider a random variable X with probability density function (pdf)

$$f_X(x) = \frac{x}{4}e^{-\frac{x^2}{8}}, \ \ x \geq 0$$

a) Write an R function to compute the above pdf.
b) Check whether this function is indeed a pdf (i.e., that it integrates 1) by:

i. A sum approximation of the form $\sum f(x_i)\Delta(x)$, where $\Delta(x)$ is a "small" increment. *Hint*: create a sequence vector (over a relatively large interval and with a small increment), evaluate your function in a) on the sequence vector, multiply the evaluation by the increment and finally sum.
ii. Using the `integrate(f, lower, upper)` function. *Note*: `$value` gives the value of the integral.

c) Compute the expected value and the variance of X.
d) Modify your function in a) so that an error message is displayed if a negative value is given as input.

38. We know that

$$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots=\sum_{n=0}^{\infty}\frac{1}{2^n}=2\,.$$

Using a `while` loop, find a value N such that

$$2-\sum_{n=0}^{N}\frac{1}{2^n}\leq\epsilon\,,$$

where $\epsilon = 0.00001$.

39. Write a function that performs the inner product of two vectors using for loops. Using `microbenchmark()` compare the performance of your implementation with the inner product using the `%*%` operator. Try with large vectors, let's say length 100, and `times = 10000`.

40. Consider the data set "EuStockMarkets" in R corresponding to the "Daily Closing Prices of Major European Stock Indices, 1991-1998".

a) Write an R program to plot a histogram of "DAX". *Note*: to access "DAX" use EuStockMarkets[, 1].
b) Compute the log-returns of "DAX". Recall that the log-returns are computed as

$$\log(P_t/P_{t-1})\,,$$

where P_t are the closing prices.
c) Compute the mean and standard deviation of the log-returns.
d) Plot a histogram of the log-returns.
e) In your plot in d), add the density function of a normal distributed random variable with mean and standard deviation the values computed in c).
f) Plot "DAX" versus "SMI" and compute their correlation.

41. Let X be a normal random variable with mean 2 and variance 9, that is, $X\sim\mathcal{N}(2,9)$. Compute:

a) $\mathbb{P}(|X|\leq 2.4)$.
b) The 95% quantile of the given distribution.

42. Consider the `cars` data set in R.

a) Compute the correlation between `speed` and `dist`, and create a scatter plot to compare `speed` vs `dist`. Do you see any relationship?
b) Fit a linear regression model to explain distance in terms of speed.
c) Add the regression line to your plot in a). *Hint:* this can be done using the `abline()` function applied to your regression model in b).
d) Predict `dist` for values of `speed` of 28 and 30.
e) Does the model seem to satisfy the assumptions of mean zero, constant variance, and normality for the residuals?

8.2 Related to Chapter 7

43. A stock price is currently 20. It is known that at the end of 1 year it will go up by 10% or down by 5%. The risk-free interest rate is 8% per annum with annual compounding. Write the R code to find the initial value of a 1-year European put option with a strike of 20.

44. A stock price is currently 40. It is known that at the end of 9 months, it will either be 45 or 35. The risk-free interest rate with continuous compounding is 10% per annum.

a) Write the R code to find the initial value of a 9-month European call option on the stock with an exercise price of 40.

b) Describe the hedging portfolio for the option.

45. A stock price is currently 30. Over each of the next two 4-month periods, it is expected to go up by 8% or down by 10%. The risk-free interest rate is 12% per annum with quarterly compounding. Write an R code to find the initial value of an 8-month European call option with a strike price of 34.

46. A stock price is currently 100. During each three-month period for the next six months, it will increase by 10% or decrease by 10%. The risk-free interest rate is 8% per annum with quarterly compounding.

a) Write an R program that gives the binomial tree evolution of the stock price.
b) Write an R code to find the initial value of a six-month European put option with strike K = 105.

47. A stock is currently priced at 30. Over each of the next two 2-month periods, it is expected to increase by 8% or decrease by 10%. The risk-free interest rate is 10% per annum with continuous compounding. What is the initial value of a 4-month derivative with strike $K = 30$ and that pays off $X = \max(S_T^2 - K, 0)$, where S_T is the stock price in four months?

48. A stock price is currently 40. In two months, it will either be 42 or 38, and during this period, the risk-free interest rate is 6% per annum with monthly compounding. If it rises to 42 in two months, then it will either be 48 or 40 after another two months. If it drops to 38 after the first two months, then after another two months, it will either be 42 or 34. The risk-free interest rate is 10% per annum with monthly compounding during the second two-month period. Write an R program to calculate the initial value of a derivative that pays off $\frac{1}{2}\left(\max(44 - S_T, 0)\right)^2$, where S_T is the stock price in four months.

49. It is known that for any $a \in \mathbb{R}$, the standard Brownian motion satisfies that $W(t) > a$ with probability 1 for some $t > 0$. Note that in the lecture notes, we simulated a standard BM over a fixed time period T. In this exercise, we let the time vary, and we fix a threshold $a > 0$, then simulate a standard BM until it reaches the given threshold. Write an R program to plot one trajectory of a standard Brownian motion until it reaches $a = 1$. *Hint*: Use a while loop to generate normal distributed increments until the condition is satisfied.

50. Via simulations, approximate the probability that an arithmetic Brownian motion with drift 0.5 and volatility 0.1 is above 1 at time 2. Compare your result with the theoretical probability.

51. Consider a European put option on a non-dividend-paying stock with current price is 40. The strike price is 38, the risk-free interest rate is 12% per annum with continuous compounding, the volatility is 30% per annum, and the time to maturity is three months:

a) Find the price of the option using the BS formula.
b) Approximate the price of the option using simulations (10000).
c) Approximate the option's price using a binomial model with 1000 steps.

52. A stock has current price $S(0) = 1416$. What is the value of an 8-month European call option on that stock with a strike price of 1450 if the volatility is 0.42 and the risk-free interest rate is 5% per annum with continuous compounding? Find the greeks for the corresponding option. Plot vega again the initial stock price $S(0)$ ranging from 0.5 to 2000.

2024 | Nuno M. Brites | nbrites@iseg.ulisboa.pt