Strategic Asset Allocation in Fixed Income Markets: A Matlab Based User's Guide

Ken Nyholm

Dcember 2007
Preliminary and incomplete version

Contents

1 Strategic Allocation **7**
- 1.1 Introduction . . . 7
- 1.2 Components of the Investment Process . . . 8

2 Essential Elements of Matlab **9**
- 2.1 Introduction . . . 9
- 2.2 Getting started[1] . . . 10
- 2.3 Introductory matrix algebra . . . 15
- 2.4 Organising data . . . 20
- 2.5 Creating functions . . . 22
  - 2.5.1 Branching and looping . . . 23
  - 2.5.2 An example of a simple function . . . 25
  - 2.5.3 Calling functions in Matlab(R) . . . 27
- 2.6 The linear regression . . . 28
  - 2.6.1 The basic setup . . . 28
  - 2.6.2 Maximum likelihood . . . 31
- 2.7 Some estimation examples . . . 32
- 2.8 A brief introduction to simulations . . . 38
  - 2.8.1 Generating correlated random numbers . . . 43

3 Fixed Income Preliminaries **47**
- 3.1 Introduction . . . 47
- 3.2 Spot rates and yields . . . 47
- 3.3 Forward rates . . . 57
- 3.4 Bond pricing functions . . . 58
- 3.5 Exercises . . . 59

[1] More information about Matlab(R) and how to use it can be found on the internet, see for example, "www.mathworks.com" and "www.econphd.net/notes"

4 Risk and Return Measures **60**
4.1 Introduction . . . 60
4.2 Risk Measures . . . 60
4.2.1 Value-at-risk and Expected Shortfall . . . 62
4.2.2 Duration and modified duration . . . 69
4.3 Fixed Income Returns . . . 77

5 Term Structure Models **82**
5.1 Introduction . . . 82
5.2 Not-Necessarily Arbitrage Free Models . . . 83
5.2.1 Nelson and Siegel . . . 83
5.2.2 Svensson and Soderlind . . . 87
5.3 Arbitrage Free Models . . . 88
5.3.1 Vasicek . . . 91
5.3.2 Multi-factor models: an example . . . 93

6 Asset Allocation **99**
6.1 Introduction . . . 99
6.2 Efficient portfolios . . . 99
6.3 Diversification . . . 108
6.4 The minimum variance portfolio . . . 112
6.5 Asset weight constraints . . . 115
6.6 The Capital Asset Pricing Model . . . 122

7 Statistical Tools **126**
7.1 Introduction . . . 126
7.2 The Vector Auto Regression . . . 127
7.2.1 Order of integration . . . 128
7.3 Regime switching models . . . 130
7.3.1 Introduction . . . 130
7.4 Yield curve models in state-space form . . . 143
7.4.1 The Nelson-Siegel model in state-space . . . 143
7.5 Importance Sampling . . . 150
7.5.1 Some theory . . . 150
7.5.2 An example . . . 151

8 Building graphical user interfaces **156**
8.1 Introduction . . . 156
8.2 The "guide" development environment . . . 157
8.3 Creating a simple GUI . . . 159
8.3.1 Plotting the yield curve . . . 162
8.3.2 Estimating λ and yield curve factors . . . 166

9 Useful Formulas and Expressions **172**
9.1 Introduction . . . 172
9.2 Matrix operations . . . 172
9.2.1 Definitions . . . 172
9.2.2 Sum . . . 173
9.2.3 Product . . . 173
9.2.4 Transpose . . . 173
9.2.5 Symmetric matrix . . . 173
9.2.6 The Identity matrix: . . . 174
9.2.7 Determinant . . . 174
9.2.8 Rank . . . 174
9.2.9 Inverse . . . 175
9.2.10 Trace . . . 175
9.2.11 Powers . . . 176
9.2.12 Eigenvalues and eigenvectors . . . 176
9.2.13 Positive definite . . . 176
9.2.14 Matrix differentiation . . . 177
9.3 Decompositions . . . 177
9.3.1 Triangular . . . 177
9.3.2 Cholesky . . . 177
9.3.3 Eigenvalue . . . 178
9.4 Basic rules . . . 178
9.4.1 Index rules . . . 178
9.4.2 Logarithm rules . . . 179
9.4.3 Simple derivatives . . . 179
9.4.4 Simple integrals . . . 180
9.5 Distributions . . . 181
9.5.1 Normal . . . 181
9.5.2 Multivariate normal . . . 181
9.5.3 t-distribution . . . 181
9.5.4 Copulas . . . 181

9.5.5 Vasicek's limiting distribution 182
9.6 Functions 182
9.6.1 Linear (affine) function 182
9.6.2 Quadratic function 183
9.6.3 General polynominals 183
9.6.4 Exponential 183
9.6.5 Logarithm 183
9.6.6 Error function 184
9.6.7 Inverse 184
9.7 Taylor series approximation 184
9.8 Interest rates, returns and portfolio statistics 185
9.8.1 Cummulative arithmetic return 185
9.8.2 Average arithmetic return 185
9.8.3 Cummulative geometric return 185
9.8.4 Average geometric return 185
9.8.5 Compounding of interest rates 185
9.8.6 Portfolio statistics 186

Preface

Initially, the work contained in this book was intended as background readings for students at the Finance Department of the Frankfurt University for a course on "Fixed Income Strategic Asset Allocation in Practice", which I started to teach some years ago. A literature search for an appropriate text book that combines fixed income analysis, portfolio theory, and in-depth computer based applications concluded unsuccessfully. Consequently, I set out to write lecture notes, aiming to illustrate by example and explain topics arising in the intersection between fixed income portfolio management and applied econometrics. As a computer tool I decided to rely on Matlab(R).

The main purpose of the book is to provide assistance to readers who are interested in the implementation of financial and econometric models in Matlab(R). While the provided annotated examples by no means claim to be "the best way" to program and implement such models and techniques, it is the hope that the text still can serve as a valuable introductory treatment relevant for students and practitioners alike. In particular, many practitioners (and students, it seems) cannot find the time (although, probably for different reasons) to do the necessary foot-work to get started implementing models in Matlab(R) - with those in mind, the aim of this book is to lower the "barrier of entry" to successful implementation of financial models.

It is my belief that a thorough understanding of financial concepts, their strengths, weaknesses and practical applicability is best facilitated by the use of concrete examples. This idea is the underpinning pedagogical philosophy of the book: All central concepts and theories are illustrated by Matlab(R) implementations, which are accompanied by detailed descriptions of the programming steps needed. In this way, the book can be used as an introduction to finance and/or Matlab(R).

The main prerequisites for reading the book are: a keen interest to explore financial models in practise and an educational background roughly equivalent to a bachelor's degree in a moderately technical field. All concepts and techniques are introduced from the basic level so the latter prerequisite is easily compensated by the former: Mathematically adept individuals will probably find the text and examples relatively easy while individuals with somewhat rusty mathematical skills (probably) will find the text refreshing.

[INSERT DESCRIPTION OF THE CHAPTERS OF THE BOOK...]

While having a different focus and coverage compared to the current ex-

position, it should be mentioned that the above referred literature search did uncover some very interesting books in the juncture of finance and Matlab(R) programing, see e.g. McLeish (2005) for a thorough description of efficient simulation methods mainly applied to the area of derivatives pricing and variance reduction techniques; Meucci (2005) gives an in-depth account of risk and asset allocation drawing on elements from the theory of multivariate statistics; Down (2005) contains a good introduction to measurement of financial risks; and Brandimarte (2002) focuses on optimisation methods useful in financial applications.

The book is accompanied by an online resource at the webpage: www.KenNyholm.com where all Matlab(R) programs referred in the text can be downloaded for free. The site also contains lecture slides that can be used by any who want to include the book, in full or in parts, in their teaching. In addition, answers to selected end-chapter exercises can be downloaded in pdf format.

This version: 17 December 2007.

Acknowledgements

Thanks to my colleagues in the Risk Management Division of the European Central Bank: Ulrich Bindseil, Matti Koivu, Evangelos Tabakis, and Julia Weber for providing comments on previous drafts. I especially want to thank Han van der Hoorn for his detailed comments, suggestions and efforts to improve the text.

Chapter 1

Strategic Allocation

1.1 Introduction

The goal of strategic asset allocation is to find an optimal allocation of funds across different asset classes subject to a relatively long investment horizon. The optimal allocation of funds should always reflect the risk return preferences of an institution and the machinery underlying the strategic asset allocation decisions should be based on a transparent and accountable process with which such allocations can be determined and reviewed at regular intervals. Often "Modern Portfolio Theory" is presented following Markowitz (1959) and Sharpe (1964) in the context of the Capital Asset Pricing Model (CAPM) and mean-variance portfolio analysis as the basic theory for how equity markets behave in equilibrium and how investors should position themselves on the efficient frontier, depending on their risk aversion. This theory is central to the understanding of modern finance and thus important for students and market practitioners alike. However, when it comes to actual portfolio allocation decisions and the practical implementation of portfolio allocation decisions in public and private investment organizations the CAPM leaves, quite understandably, many questions unanswered. It is some of these missing answers that we address in this book. In doing so, the viewpoint of a strategic investor is taken, however, elements relevant for tactical asset allocation and portfolio managers are also touched upon. In particular, the focal point of the exposition is that of a fixed income asset manager. This perspective naturally narrows the investment universe considerably. As a consequence, the topics treated in the book revolve mainly around an invest-

ment universe comprising fixed income securities.

1.2 Components of the Investment Process

A list of the topics covered in this section

- Strategic asset allocation
  - Purpose / general properties: yard stick for active management; reflects the institutions long-term risk return preferences; replicability; review frequency; stability of modified duration over time; rebalancing; choice of instrument universe; relevant literature.
  - How to generate the asset allocation proposal; decision making process in investment houses; asset constraints; in-stability of Markowitz optimisation / parameter uncertainty; utility functions; existence of a risk-free asset
- Tactical asset allocation decisions
  - Purpose / general properties: outperform strategic allocation; How and by the use of which means? Limits to outperformance (it is debatable whether it add value); objective function;
  - Techniques and tools: Black-Litterman / Bayesian approaches; performance evaluation tools;

[To be written...]

Chapter 2

Essential Elements of Matlab

2.1 Introduction

This chapter illustrates some core functionalities of Matlab(R) using easy-to-understand examples taken from the areas of Matrix Algebra and Econometrics - so, this section can also be used as a brief recap of these areas. First, a review is given of the programming and matrix capabilities of the programming package; then it is shown how Matlab(R) can be used to solve the linear regression problem; and finally, the chapter concludes by showing examples of how simulation studies can be undertaken by the use of Matlab(R). Throughout the book Matlab(R) will be used in solving financial problems of an empirical nature. Hence, in later chapters, there will be ample opportunity to revisit and expand on the topics introduced in this introductory chapter.

Learning objectives

- Get familiar with Matlab(R)
  - help function
  - graphing functions
  - assign values to vectors and matrices in two and three dimensions
  - storing data in structured variables
  - mathematical operators

  - loops and conditioning

- Review central concepts from the areas of matrix algebra and quantitative techniques
  - Matrix algebra
    * vector and matrix operations
    * transpose of vectors and matrices
    * square matrices
    * matrix inversion
  - Quantitative techniques
    * linear regression
    * likelihood estimation
    * simulation techniques

2.2 Getting started[1]

It is assumed that Matlab(R) is already installed and fully functional. The command prompt ">>" is one way that you as a user can work with Matlab(R). Lets start simple by drawing a graph of random numbers using command prompt input. Note that through out the text, for notational purposes, Matlab(R) examples are contained in a box as seen below. The command prompt ">>" is omitted in these examples but line numbers are added to facilitate annotation. These line number should naturally not be entered when trying out the examples!

```
[1] r = randn(10,1);              % assigns 10 N(0,1) random values to vector r
[2] plot(r,'ko')                  % plots r with black 'o' marking
[3] xlabel('Observation number')  % adds x-axis text
[4] ylabel('Value')               % adds y-axis text
```

[1] More information about Matlab(R) and how to use it can be found on the internet, see for example, "www.mathworks.com" and "www.econphd.net/notes"

In the above example, line [1] generates a column vector r that contains 10 normally distributed random numbers. To this end the built-in random number generator *randn* is used. This function requires two inputs that determine the dimension of the output matrix/vector. The first input determines the number of columns, and the second determined the number of rows. Random numbers do not necessarily have to be normally distributed, for example, Matlab(R) can also generate uniformly distributed random numbers: this is done by the function *rand*. Line [2] plots r, and lines [3] and [4] adds axis labels to the plot. Two special characters are used in the above example; these are:

- "%" indicates to Matlab(R) that what ever follows after should not be processed. Hence, the % character can be used in Matlab(R) to write help-text to the users and colleague Matlab(R) programmers. Additionally, a collection of lines preceded by "%" at the very beginning of a Matlab(R) function constitutes the help-text of that function and is printed to screen if a "help my_function_name" command is invoked.
- ";" tells Matlab(R) not to make a screen-print of the answer produced by the calculations conducted by the line concluded by ";".[2]

Executing the above lines of code generates a picture similar to Figure 2.1. When executing the lines from the above example, you might get something which is not exactly equal to the graph shown. This is because we draw random numbers, and every draw will be different, due to the randomness. More information about the random number generator in Matlab(R) can be obtained via the built-in help function: simply type:

```
[1] help randn
```

and Matlab brings up the following message (only part of the actual help message is reprinted below):

[2] Try executing the command:

```
[1] >> r = randn(5,1)
```

i.e. omitting the ";".

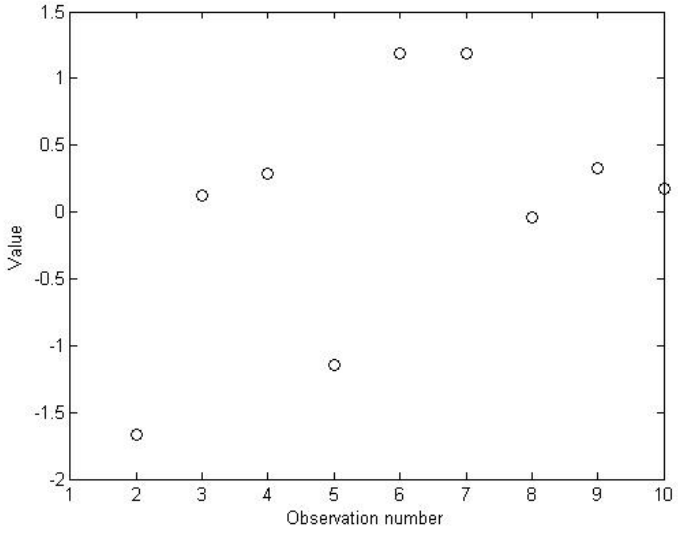


Figure 2.1: Example of the "plot" command

```
RANDN Normally distributed random numbers.
R = RANDN(N) returns an N-by-N matrix containing pseudo-random values drawn
from a normal distribution with mean zero and standard deviation one.  RANDN(M,N)
or RANDN([M,N]) returns an M-by-N matrix.  RANDN(M,N,P,...)  or RANDN([M,N,P,...])
returns an M-by-N-by-P-by-...  array.  RANDN with no arguments returns a
scalar.  RANDN(SIZE(A)) returns an array the same size as A.
```

This is a general feature of Matlab(R): for every Matlab(R) function you can see the accompanying help text by using the "help" command. In this connection it should also be mentioned that a searchable help function exists. This one can be activated by the following command:

```
[1] helpdesk
```

Returning to the plotting example from above. Situations might arise where it is desirable to show several graphs in one plot-window, different graphs in different plot-windows at the same time, or different graphs in different plot-windows but collected in one display. To this end the commands "hold

on", "figure" and "subplot" are helpful; the examples below demonstrates this.

```
[1] clear all                  % deletes all variables in the workspace
[2] clc                        % clears the view on the workspace
[3] r1 = rand(7,1);            % generates uniform random variables
[4] r2 = [5;4;3;0;-1;-1;-10];  % column vector of the entered values
[5] plot(r1)
[6] hold on
[7] plot(r2)
```

These lines of code generate the graph in Figure 2.2.

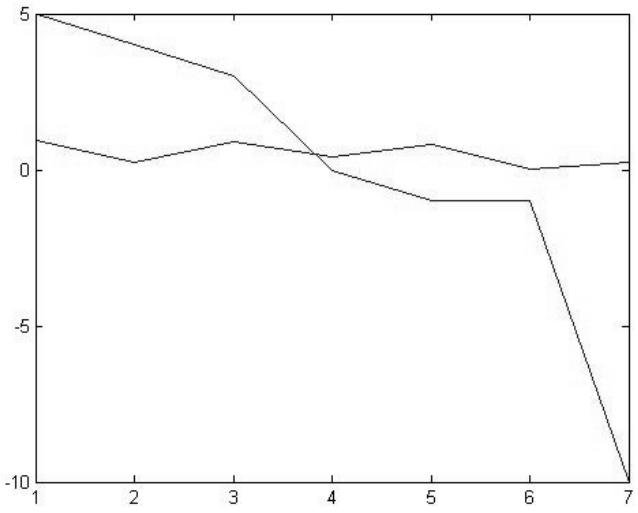


Figure 2.2: Example of the "hold on" command

Delete the plot, as usual in Windows(R), by clicking in the right hand corner of the plot-window. Then proceed with the following:

```
[1] plot(r1)
[2] figure
[3] plot(r2)
```

These lines of code plot the vectors $r1$ and $r2$ in two separate windows (not shown here).[3] If the "figure" command is omitted, i.e. two (or more) plot statements are repeated right after each other, Matlab will print all graphs in the same window and only the last plot remains visible.

The last plotting example below shows how four graphs can be displayed together in one plot by using the "subplot" command. Close any open figures and proceed with the example below.

```
[1] clear all
[2] clc
[3] r1 = rand(10,1); r2 = rand(10,1); r3 = randn(10,1); r4 = randn(10,1);
[4] whos                        % displays the variables on the workspace
[5] subplot(2,2,1), plot(r1)
[6] subplot(2,2,2), plot(r2)
[7] subplot(2,2,3), plot(r3)
[8] subplot(2,2,4), plot(r4)
```

These lines generate a figure similar to the one shown below.[4]

There exist basically three ways in which the user can communicate with Matlab(R), of which two have been illustrated above:

- Via the command line: i.e. writing a command at the Matlab(R) command prompt ">>".
- Via a function written in a file and called from the command prompt. This technique will be illustrated below.
- Via a script file. A script file is a series of commands collected in a separate file that can be executed from the Matlab(R) command prompt. The practicality of a script file is that it invokes many Matlab(R) commands after each other and it can include function calls as well. In this way, a script serves as a useful way to bundle and keep track of commands sent to Matlab.

[3] More about vectors and matrices follow in Section 2.3.

[4] The "whos" command used in line [4] in the above is naturally not needed to generate the example plots.

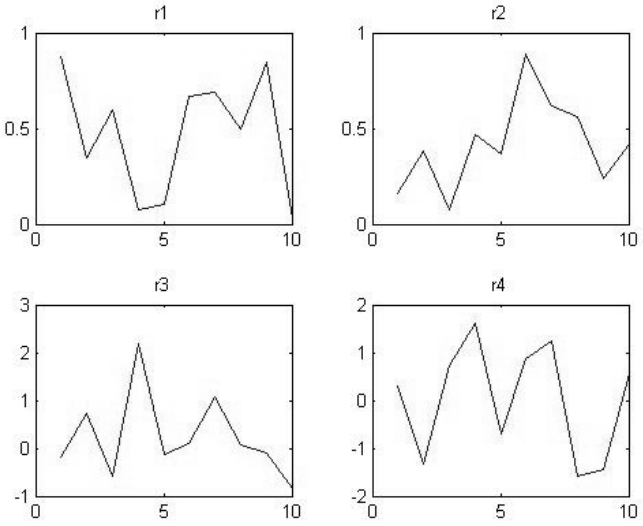


Figure 2.3: Example of the "subplot" command

2.3 Introductory matrix algebra

This section reviews some basic matrix operations as well as how they are implemented in Matlab(R). The presentation draws on Lipschutz and Lipson (2001) and Hamilton (1994)[Appendix 4].

To distinguish vectors and matrices more easily let uppercase letters represent matrices, and lower case letters represent vectors. Hence, C would be a matrix while c would be a vector. The superscript T is used for the transpose of a matrix or a vector, for example, C^T and c^Twould be the transpose of the before mentioned matrix and vector. The transposition operator reorganises the rows and columns of vectors and matrices: what was before a column becomes a row after the transposition is invoked. Subscripts, in connection with matrices and vectors are used to report the dimension of the object at hand, e.g:

$$C = C_{(n,k)} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,k} \\ \vdots & & \vdots \\ c_{n,1} & \cdots & c_{n,k} \end{bmatrix},$$

in the case of a matrix, where n gives the number of rows and k the number of columns. Hence a column vector will have $k = 1$ and a row vector will have $n = 1$. Effectively, C^T is:

$$C^T = C^T_{(k,n)} = \begin{bmatrix} c_{1,1} & \cdots & c_{n,1} \\ \vdots & & \vdots \\ c_{1,k} & \cdots & c_{n,k} \end{bmatrix}.$$

A special matrix is denoted the "identity" matrix and is defined by:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Similarly, the symbol "$\mathbf{1}$" is used to denote a vector of ones, so $\mathbf{1} = [1, 1, \cdots, 1]^T$. Mathematical operators are used in their usual guise, however, in the spirit of Matlab, a "." in front of an operator signifies an element-by-element operation. The example below illustrates the difference between element-by-element multiplication and conventional multiplication.

```
[1] Q=[1 2 3; 4 5 6; 7 8 9];
[2] q=[1;1;1];
[3] z1=Q*q          % regular matrix multiplication
[4] z2=Q(:,1).*q    % element-by-element multiplication
```

The result stored in $z1$ is equal to:

$$z1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} (1*1)+(2*1)+(3*1) \\ (4*1)+(5*1)+(6*1) \\ (7*1)+(8*1)+(9*1) \end{bmatrix} = \begin{bmatrix} 6 \\ 15 \\ 24 \end{bmatrix},$$

i.e. following the general matrix multiplication rule

$$z = \begin{bmatrix} Q_{1,1} & Q_{1,2} & Q_{1,3} \\ Q_{2,1} & Q_{2,2} & Q_{2,3} \\ Q_{3,1} & Q_{3,2} & Q_{3,3} \end{bmatrix} \begin{bmatrix} q_{1,1} \\ q_{2,1} \\ q_{3,1} \end{bmatrix} = \begin{bmatrix} (Q_{1,1}*q_{1,1})+(Q_{1,2}*q_{2,1})+(Q_{1,3}*q_{3,1}) \\ (Q_{2,1}*q_{1,1})+(Q_{2,2}*q_{2,1})+(Q_{2,3}*q_{3,1}) \\ (Q_{3,1}*q_{1,1})+(Q_{3,2}*q_{2,1})+(Q_{3,3}*q_{3,1}) \end{bmatrix},$$

noting that each row of Q is multiplied by the vector q. The dimension of the result is thus determined by the dimensions of the inputs; in our example

$$Q_{(3,3)} * q_{(3,1)} = z_{(3,1)},$$

and more generally

$$Q_{(m,n)} * W_{(n,k)} = Z_{(m,k)}.$$

Hence, matrix and vector dimensions show directly whether an operation is possible, and what the dimension of the result is going to be.

Let's look at the calculation of $z2$ in the example. Here the element-by-element multiplication is used, so each element in Q is multiplied by the same entry in q. Since the dimension of q is $(3,1)$ we can not perform element-by-element multiplication for all elements in Q, because it has dimension $(3,3)$.[5] Therefore it is only possible to perform the desired calculation for a partition of Q or by increasing the dimension of q. In the above example the former alternative is chosen (what one chooses to do is naturally dictated by the purpose of the calculation - the discussion here reflects that the calculations relate to a purposeless example), by performing the calculations for the partition of Q constituting its first column. A partition of Q constituting its first column is generated by writing: $Q(:,1)$, where ":" refers to all elements, and 1 to the column number. In other words, the following calculation is performed:

$$z2 = Q(:,1) .* q = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} .* \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1*1 \\ 4*1 \\ 7*1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}.$$

Naturally, it is also possible to add and subtract matrices as long as their dimensions allow it. It is hence a requirement for matrix addition and subtraction that the involved matrices have an identical number of rows and columns, i.e. if we have two matrices $A_{(m,n)}$ and $B_{(k,l)}$ it is necessary that $m = k$ and that $n = l$. The following example illustrates this:

[5]Note, however, that other software packages, e.g. Gauss lets you perform this type of element-by-element multiplication even if the dimensions do not match. This is done because the software package is pre-programmed to assume that you want to do the calculation for all elements and thus automatically expands the dimension of q to match that of Q. In our example this could be achieved by constructing $\widetilde{q}$ as a matrix where q would be repreated 3 times. This can be achieved in Matlab by e.g. using the "repmat" command.

```
[1] A = [1 2 3; 4 5 6];
[2] B = [ 2 2; 3 3];
[3] A+B
???  Error using ==> plus
Matrix dimensions must agree.
```

whereas

```
[1] A = [1 2 3; 4 5 6];
[2] B = [ 2 2 2; 3 3 3];
[3] A+B
ans =
3 4 5
7 8 9
[4] A-B
ans =
-1 0 1
1 2 3
```

Matrix addition and subtraction functions at the level of the individual elements of the matrices akin to the element-by-element multiplication operator ".*" seen above. There are a number of other matrix rules that can come in handy, these are:

- $(AB)^T = B^T A^T$
- $AB \neq BA$
- if $(A^T) = A$, then A is said to be symmetric, i.e. is a mirror image around the diagonal

These rules are illustrated by examples. One of the issues mentioned above is that the feasibility of matrix operations can, to some extent, be evaluated on the basis of the dimensions of the matrices and vectors involved. The first bullet is a point in case: if for example A is of dimension (m, n) $\left[\text{i.e. } A_{(m,n)}\right]$ and B is of dimension (n, k) $\left[\text{i.e. } B_{(n,k)}\right]$, then the product of $AB = C$ will be of dimension (m, k). Now, consider the transposition $(AB)^T = C^T$ which will have dimension (k, m). One way to create a result having this dimension

is to perform the calculation: $B^T_{(k,n)} * A^T_{(n,m)}$ which exactly has dimension (k, m), as it should have. This intuition can be further helped by example calculations in Matlab(R).

```
[1] A = [1 2 3; 4 5 6;7 8 9];
[2] B = [ 2 2; 3 3; 4 4];
[3] C=(A*B)
C =
20 20
47 47
74 74
[4] C'
ans =
20 47 74
20 47 74
[5] B'*A'
ans =
20 47 74
20 47 74
```

It is noted that in Matlab(R) the sign used to transpose a variable is ', so our text-notation: B^T becomes: `B'` in Matlab(R).

An argumentation similar to the one above shows the correctness of the second bullet point, namely that $AB \neq BA$. For non-square matrices, this proposition can be confirmed simply by looking at the dimensions of the two sides of the non-equality sign. For square matrices we show it by example:

```
[1] A = [1 2 3; 4 5 6;7 8 9];
[2] B = [ 2 2 2; 3 3 3; 4 4 4];
[3] C=A*B
C =
20 20 20
47 47 47
74 74 74
[3] D=B*A
D =
24 30 36
```

```
36 45 54
48 60 72
```

In Matlab(R) it is possible to work with partitions of matrices. In the example above a partition of Q was taken, namely the first column. Other partitions are made in similar ways. It is also possible to refer to individual elements of matrices or clusters of elements by specifying the column and row numbers that you are interested in. The following examples illustrate this.

```
[1] Q = [1 2 3; 4 5 6; 7 8 9];
[2] r1 = Q(2,:);      % selects the second rows of matrix Q
[3] r2 = Q(3,2);      % selects the 3rd row and 2nd column element
[4] r3 = Q(1:2,3);    % selects the 1st and 2nd elements in column  3
```

The results of this example, given the matrix Q, are:

$$r1 = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

$$r2 = 8$$

$$r3 = \begin{bmatrix} 3 \\ 6 \end{bmatrix}.$$

2.4 Organising data

It is sometimes useful to store the results generated by Matlab(R) in a compact fashion, either when organising the input and output to and from functions[6], when running simulation studies, or simply as a convenient way of organising data. As hinted in the learning objectives in the introduction to this chapter, there are several ways of doing this. Here two such ways are illustrated: (i) in a structured variable and (ii) in a 3-dimensional matrix.

Concerning (i), a structured variable is an extremely powerful construct akin to a tree-structure, where the trunk of the tree is the name of the structured variable and the branches and leaves of the tree are sub-variable

[6]This will be demonstrated later on.

names and values. Imagine that we want to organise the data used in the above example in a structure variable denoted by *Sum_data*. Then variable names and values can be added to this structure in the following way:

```
[1] Q = [1 2 3; 4 5 6; 7 8 9];
[2] r1 = Q(2,:);
[3] r2 = Q(3,2);
[4] r3 = Q(1:2,3);
[5] Sum_data.Q = Q;
[6] Sum_data.r1 = r1;
[7] Sum_data.r2 = r2;
[8] Sum_data.r3 = r3;
```

It is now possible to investigate the content of the structured variable *Sum_data* by writing:

```
[1] Sum_data
```

Invoking the above command displays the variables contained by the structure named *Sum_data*.

As a complement to storing data in a 3-dimensional matrix it is possible to use an indexed structured variable. Say, for example, that a matrix of output values is generated many times, as it is often the case in simulation studies, and let j be a counter-variable that tracks the simulation number. The following example illustrates the use of 3-dimensional matrices and indexed structures as output containers. The choice of method depends on individual preferences, although the use of indexed structure variables is more flexible because the dimension of the data matrix that is being stored does not necessarily have to be identical from iteration to iteration.

```
[1] clear all;
[2] for (j=1:10)
[3]     rdim = floor(rand()*10)+1;
[4]     ex_dat1 = randn(rdim,5);
[5]     ex_dat2 = randn(5,5);
[6]     struct.A{j} = ex_dat1;
[7]     Mat3D(:,:,j) = ex_dat2;
[8] end
```

Naturally, the above example is of little practical use, however it serves the purpose of illustrating the different options that can be used to store information generated while conducting e.g. simulation studies. It is a matter of individual preferences which method to choose if the dimension of the output matrices is identical from iteration to iteration. If the dimension varies from iteration to iteration, the simplest choice is to use the structured variable.

In the above example, the structure variable "*struct*" has just one variable attached to it, called "A". In principle, we could thus have used "struct" by itself, but for illustrative purposes an additional level is added to the structure. 10 iterations are conducted as determined by line [2], hence the counter variable j runs from 1 to 10. Line [3] defines the dimension of the rows of the output matrix as a random number between one and 10. The function $floor(z)$ rounds the elements of z downward to the nearest integer; the mirror image of $floor(z)$ is $ceil(z)$ which integer-rounds the elements of z upward: in effect, $floor(1.5) = 1.0$ and $ceil(1.5) = 2$. It is recalled that the function $rand()$ generates uniformly distributed random numbers between 0 and 1, while $randn()$ generates normally distributed random numbers. Line [4] allocates the matrix of random numbers, of a random row-dimension, to the variable ex_dat1. Line [5] generates further output, however, here the dimension is fixed to be five-by-five. Lines [6] and [7] store the output information is a structure and a 3-dimensional matrix, respectively. It is now possible to inquire the content of the output matrices. For example, invoking the command "$struct.A\{4\}$" and "$Mat3D(:,:,4)$" will print the content of the 4'th iteration. In particular, the 3-dimensional "container" is organised as follows: $Mat(nRows, nCols, nDepth)$, where $Mat(nRows, nCols)$ correspond to a normal (2-dimensional) matrix having $nRows$ rows and $nCols$ columns. The third entry $nDepth$ refers to how many of the 2-dimensional matrices that are stored next to each other.

2.5 Creating functions

All the calculations performed above were done in "command line" mode. This means that expressions were created and sent one-by-one to Matlab(R) and executed in the order they were sent. Using Matlab(R) in this way is like having a very powerful pocket calculator. Similar to the functions that can

be written and stored in more advanced pocket calculators, Matlab(R) also allows for programs to be created: in Matlab(R) terminology such programs are called *functions*. Functions are for example useful when a given task has to be performed several times. Rather than writing the instructions to Matlab(R) as individual lines, as one would do in command line mode, the instructions can be collected in a function.

Before writing a function it is necessary to look at some of the building blocks that can be used to create functions. These are used to do conditional branching, i.e. to do something specific if certain criteria are fulfilled, and to loop, i.e. to repeat a set of functions a given number of times.

2.5.1 Branching and looping

Branching is needed when a given set of instructions are to be performed only if certain criteria are fulfilled. Functions and programs/scripts can be branched by the use of "if" and "case" statements. "case" is useful in the event that a given known number of possible branches exist, while "if" is a more universal branching structure. The generic forms of these statements can be seen in Matlab(R) by typing:

```
[1] help if
```

and

```
[1] help case
```

For convenience, they are repeated below along with the relation operators:

Operator	Explanation
<, <=	Less than, Less than or equal to
>, >=	Greater than, Greater than or equal to
==	Equal to (not to be confused with "=" that assigns a value)
~=	Not equal to
~	Not

The "if" statement has the following structure:

```
if expression
  statements
elseif expression
  statements
else
  statements
end
```

and the "case" statement has this structure:

```
switch expression
case expression_A
  statements
case {expression_B, expression_C}
  statements
otherwise
  statements
end
```

Looping can be done in Matlab(R) by the use of "for" and "while" statements. The former statement is often used when a set of statements are to be executed a known number of times, and the "while" statement is used to loop as long as a given set of criteria are fulfilled. The generic structure of these statements are given below.

"for" structure	"while" structure

for (variable=1:step_size:N)
  statements
end
	while (variable evaluated against expression)
	  statements
	end

In the "for" structure the "variable=1:step_size:N" denotes the counting part of the loop, and should be read: for variable equal one to N in steps of step_size. As seen above the default value for "step_size" is one, and the "step_size" can then be omitted if the user wants to apply a value identical to one, as it is seen in the examples above.

2.5.2 An example of a simple function

A function typically consists of three parts: the header, which specifies the name of the function, the input and output arguments, and the keyword "function", which tells Matlab(R) that the following is a function; an explanation of how to use the function and what the function does; and finally, the instructions that together define the function. Below is an example of a simple function that generates a plot of random numbers, where the user specifies the dimensions of the data to be included in the plot.

At the Matlab(R) prompt type:

```
[1] edit
```

This opens the Matlab(R) editor window that can be used to generate functions and to store them. Make a directory on the harddisk where you want to store your programs and functions and remember to include this folder in the Matlab path. One way to do this is to type:

```
[1] pathtool
```

at the command line. This allows you to browse and add the desired path(s) using a menu-driven tool. Path(s) can naturally also be added directly from the command prompt.

We are now ready to make a function. In the edit window type the following and store it in the directory you have made and added to the Matlab(R) search path. Always store the function in a file that has the same name as the function: in our case that is "EX_plot.m"[7].

```
[1] function [] = EX_plot(zz,yy,xx)
[2] %
[3] % Function that plots 2 and 3 dimensional graphs
[4] % on the basis of user input
[5] %
[6] % input
[7] % zz - data to be plotted (nObs-by-nSeries)
```

[7] As above, the numbers in brackets to the left in the function text should not be typed. These represent line-numbers and are shown to the very left in the Matlab(R) editor window.

```
[8] % yy - y axis data (nObs-by-1, or left empty for 2D plot)
[9] % xx - x axis data (nSeries-by-1)
[10] %
[11] if ( nargin==3 )
[12]     figure
[13]     surf(xx,yy,zz)
[14] elseif ( nargin==2 )
[15]     figure
[16]     plot(yy,zz)
[17] else
[18]     disp('please provide data for plotting')
[19] end
```

A short description of the function *EX_plot* is provided: line [1] contains the name of the function, and the input arguments zz, yy, xx; these are given in "reverse" order to facilitate plotting of both 2D and 3D graphs. The function does not return any value to the user; this is seen by the empty output argument [] after the *function* keyword. Lines [2]-[10] contain the header of the function, which outlines how the function can be used, what the inputs are and what the output is. Each line in the header part is preceded by the "%" character. This indicates, as mentioned in Section 2.2, that these lines constitute user-help text which will be printed if a "*help EX_plot*" command is invoked at the Matlab(R) command line. Lines [11]-[19] constitute the main part of the function. Line [11] checks if exactly three input arguments are supplied by the user. The command "*nargin*" is a Matlab(R) built-in function that counts the number of input arguments. In our example the user can specify three or two inputs, depending on whether a 3D or a 2D graph is supposed to by printed. If three arguments are supplied, and if they are supplied in the correct order, then a 3D plot will be made, following line [13] using the built in Matlab(R) function for surface plotting: *surf*. Here, "the correct order" refers to the ordering of the input so that the dimensions match the help-text of the function. Line [12] opens an empty figure window. This only ensures that the function does not overwrite an existing plot if such a plot is open at the time the function is executed. Line [14] checks if two input arguments are provided, and lines [15]-[16] subsequently constructs a 2D plot. If less than two inputs are given, line [18] prints an error message to the user.

Functions can also be defined as "in-line functions", i.e. within a script that does more than just defining the function of interest. This can save space and when applicable it facilitates generation of more compactly written code. An example of an in-line function is given below:

```
[1] fx = @(k,a,b,x) k+a*x+b*x.^2;
[2] x  = (-10:0.1:10)';
[3] h  = fx(2,0.3,0.5,x);
[4] plot(x,h);
```

Line[1] defines an in-line function of three parameters k, a, b, and the dataset x. The function is called fx and @ is the keyword that makes Matlab(R) understand that the user now wishes to specify an in-line function. Once the function is defined, either via the command line window or via a script, it can be called with different arguments. Note that line [1] uses the element-by-element squaring, i.e. the power function "^" is preceded by a "full-stop" sign to form ".^". Line [2] defines the x-variable and line [3] calls the function fx with some example arguments and stores the outcome in the vector h. Line[4] plots h against x.

2.5.3 Calling functions in Matlab(R)

There are basically three ways in which a user defined function can be used in Matlab(R). Firstly, it can be called from the command prompt to generate the desired output. This is done in the following way:

$$>> [outputs] = function\_name(inputs).$$

In this case the structured variable or the output arguments contained in [outputs] will exist in the Matlab(R) workspace. An example of this is the execution of the above code for the generation of 2D/3D plots. This function can be called from command line by invoking:

$$>> [] = \texttt{EX\_plot(zz,yy,xx)}$$

or by

$$>> \texttt{EX\_plot(zz,yy,xx)}$$

the latter is possible because the function only generates a figure plot and does not return any output arguments. Secondly, a user defined function

can be called from within another user defined function, program or script file. Calling a function in this way is similar to invoking it from command line mode; the difference is that in the former case the output arguments of the function can be used directly within the function, program or script from which it is called. Thirdly, a user defined function can be called from a built-in Matlab(R) function. For example, if a user has written a function that calculates the value of a given model conditional on a given set of parameters, and has an interest in estimating the values for the parameters that maximises or minimises this function, then the function can be called from Matlab's built-in function-minimisation routine $fmincon$. In the following chapters we will see numerous examples of this. Here only an appetiser is given by showing the two function "handles" that Matlab acknowledges. Assume that the function of interest is called my_fun, defined either as an in-line function or as an external function, then this function can be passed to $fmincon$ in either of these ways:

$$[...] = fmincon('my\_fun', ...)$$

or

$$[...] = fmincon(@my\_fun, ...).$$

2.6 The linear regression

This section shows how the concepts presented above can be used in practice. Two examples are presented: the first concerns the solution to the ordinary least squares (OLS) regression calculated by matrix algebra and the second shows how the same results can be obtained via a function that minimises the sum of squared residuals.

2.6.1 The basic setup

A linear relationship between variables, one variable y depending on other variables x_is, can be formulated as:

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + ... + \varepsilon.$$

or when collected in matrix form:

$$y_{(n,1)} = X_{(n,k)} * b_{(k,1)} + \varepsilon_{(n,1)}, \tag{2.1}$$

where $X = [1\ x_1\ x_2\ \ldots]$ collects the original data and 1 is a vector of ones that accounts for the constant in the regression. ε is an error term, that is the part of Y that cannot be explained by the right-hand-side variables. For the linear regression to produce reliable estimates,[8] it is, among other things, required that:

- the expected value of the error term is 0; otherwise the intercept term will be biased.
- the error terms are all drawn from the same distribution and have similar variances and are not correlated with one another; otherwise the variance of the parameter estimates will be biased, i.e. the variance of β will be producing unreliable inference about the estimated parameters.
- the independent variables collected in X are not linearly related, i.e. no exact linear relationship can exist between them; otherwise the problem of multicollinearity prevails and it affects the standard errors on β.

Note that β is used to represent the estimate of the true b, i.e $E[\beta] = b$. The solution to equation (2.1) by Ordinary Least Squares (OLS) proceeds in the following manner:

$$\varepsilon = Y - Xb.$$

The objective is to minimise the sum of squared residuals, i.e to minimise $e^T e$, where e is the empirical counterpart of ε:

$$\begin{aligned} e^T e &= (Y - X\beta)^T (Y - X\beta) \\ &= \left(Y^T - \beta^T X^T\right)(Y - X\beta) \\ &= Y^T Y - Y^T X\beta - \beta^T X^T Y + \beta^T X^T X\beta, \end{aligned}$$

since $Y^T_{(1,n)} * X_{(n,k)} * \beta_{(k,1)}$ is a scalar, and a scalar is equal to its transpose, we have that $\left(Y^T X\beta\right)^T = \beta^T X^T Y$, so:

[8] Reliable in this context mean BLUE, i.e. Best Linear Unbiased Estimates.

$$e^T e = Y^T Y - 2\beta^T X^T Y + \beta^T X^T X \beta.$$

To find the minima we differentiate with respect to β, set the result equal to zero, and solve for β.

$$\frac{\partial \left(e^T e\right)}{\partial \beta} = 0 - 2X^T Y + 2X^T X \beta = 0,$$

which implies that:

$$X^T X \beta = X^T Y.$$

When it is assumed that the columns of X do not exhibit exact linear relationships, i.e. when X has full rank, the inverse of $\left(X^T X\right)$ exists and β can be found as:

$$\beta = \left(X^T X\right)^{-1} X^T Y.$$

A short-hand way to obtain this relationship is by pre-multiplying (2.1) by X^T and substituting in the estimates of ε and b:

$$X^T Y = X^T X \beta + X^T e,$$

since $E\left[X^T e\right] = 0$, we obtain:

$$X^T Y = X^T X \beta,$$

and

$$\beta = \left(X^T X\right)^{-1} X^T Y.$$

It is now clear how the values for β can be calculated. It is, however, also interesting to see how precise these estimates are, once actual data is used. To this end the variance-covariance matrix of β has to be derived. Since $Y = Xb + e$, we can write:

$$\begin{aligned} \beta &= \left(X^T X\right)^{-1} X^T Y = \left(X^T X\right)^{-1} X^T \left(Xb + e\right) \\ &= \left(X^T X\right)^{-1} X^T X b + \left(X^T X\right)^{-1} X^T e \\ &= b + \left(X^T X\right)^{-1} X^T e, \end{aligned}$$

so,

$$\beta - E[\beta] = \left(X^T X\right)^{-1} X^T e,$$

and squaring this result generates the variance-covariance matrix for the estimated parameters:

$$\begin{aligned} var[\beta] &= E\left[(\beta - E[\beta])(\beta - E[\beta])^T\right] \\ &= E\left[\left(X^T X\right)^{-1} X^T e e^T X \left(X^T X\right)^{-1}\right] \\ &= \left(X^T X\right)^{-1} X^T \sigma^2 I X \left(X^T X\right)^{-1} \\ &= \sigma^2 \left(X^T X\right)^{-1}. \end{aligned}$$

The error term variance σ^2 is unknown in most practical applications and thus needs to be approximated/estimated. This is usually done in the following way:

$$\widehat{var}[\beta] = s^2 \left(X^T X\right)^{-1} = \frac{u^T u}{N-k} \left(X^T X\right)^{-1},$$

where, $u = Y - X\beta$, i.e. using the actual residuals obtained from data to approximate the error term variance. N is the total number of observations and k the number of estimated parameters.

2.6.2 Maximum likelihood

In real life the estimation challenges that one faces are typically more complex than the OLS regression illustrated above. To handle such situations the likelihood principle can be used. A general idea here is that the residuals from the model are assumed to be generated by a known distribution, e.g. a normal distribution, and by solving the chosen density function for the population parameters in an iterative fashion, using an optimiser function, the desired parameter estimates of the model and the distribution can be found. To illustrate, below we sketch how the OLS regression can be solved using the maximum likelihood (ML) estimation technique.

$$Y = X\beta + u,$$

the residuals are assumed to be normally and identically, independently distributed (Niid), which ensures comparability to the previously derived analytical solution to the OLS regression:

$$u \sim N\left(0, \sigma^2 I\right),$$

and we therefore need the normal density:

$$L\left(s^2, \beta\right) = \left(2\pi s^2\right)^{-T/2} \exp\left[-\frac{1}{2s^2}\sum_{j=1}^{T}\left(y_j - x_j\beta\right)^2\right].$$

Functions that involve *exp* are non-linear and are hence difficult to optimise. It is much easier to optimise $\log(L)$ instead of L, and this is legitimate because the parameters that optimise L also optimise $\log(L)$. For these reasons likelihood optimisation involves the log likelihood function:[9]

$$\log L = -\frac{T}{2}\ln\left(2\pi s^2\right) - \frac{1}{2s^2}\sum_{j=1}^{T}\left(y_j - x_j\beta\right)^2.$$

To put the principle into practice one needs a function that calculates the residuals and the following log likelihood function value, and a software that can minimise the function in an iterative fashion conditional upon the set of parameters describing the problem at hand, i.e. (s^2, β). Such an "optimisation" approach is also useful if parameter constraints are applied, e.g. if financial theory stipulates a certain relationship between the variables included in the regression. It is often relevant to test such parameter constraints empirically. An analytical solution to the OLS regression will quickly become quite involved, if possible at all, in the face of such constraints; while the empirical approach described above can easily accommodate constraints on individual parameters and groups of parameters.

2.7 Some estimation examples

In this section we use the two estimation principles outlined above. The data are contained in the Excel workbook called "Data". We are going to use one of the interest rate series contained in the sheet 'Rates'. Load the data into the Matlab(R) workspace by invoking:

```
[1] R = xlsread('[PATH]\Data','Rates');
```

[9] Matlab(tm) uses the function *log*(*x*) to denote the natural logarithm of x.

[PATH] refers to the directory path where the data is located e.g. 'C:\myfolder'. In order to construct the variables to be used in the regression, the matrix R needs to be modified. The first row of R contains the maturities at which the rates are observed. In the current example we focus only on the 3 months rate, i.e. on the first column of R and hypotheses the following relationship.[10]

$$r_t = m + a * r_{t-1} + b * r_{t-2} + c * r_{t-3} + u_t.$$

As can be seen from the regression equation the right-hand-side variables are all lagged versions of the left-hand-side variable. Such a regression is also called an autoregression, which is defined by the number of lags that are included. In our case three lags are included and the regression is hence termed an autoregression of order 3, in short AR(3). Naturally this concept generalises to n lags, and hence to an AR(n), and to k variables, in which case the regression is called an vector autoregression, in short VAR.[11]

To construct the left- and right-hand-side variable from R the following can be invoked:

```
[1] r     = R(2:end,1);    % allocates the 3m rate to variable r
[2] rLag3 = r(1:end-3,:);  % creates the 3.lag
[3] rLag2 = r(2:end-2,:);  % creates the 2.lag
[4] rLag1 = r(3:end-1,:);  % creates the 1.lag
[5] Y = r(4:end,:); % adjusting r
[6] X = [ ones(length(Y),1) rLag1 rLag2 rLag3 ];
```

It is worth remembering that the data is ordered so that the last observation is the most recent one in calendar time and the first observation is the oldest observation in calendar time. Given this, the above generates the needed lagged variables. If data were in reverse order then the lagging of variables would have to be reversed as well. The vector of *ones* included in line [6] accounts for the constant in the regression. Armed with the variables, the only remaining task is to apply the OLS formulas derived above. This is done below:

[10] It should be emphasised that the shown regression serves as an example only. It is not claimed that the specification is optimal in any sense or even that it fulfils statistical requirements that adhere to linear regressions in general. No tests are conducted to corroborate the chosen specification.

[11] The VAR abbreviation should not be confused with value-at-risk (VaR).

```
.....................................................................
[1] B_hat1 = pinv(X'*X)*X'*Y;   % calc the beta estimates
[2] B_hat2 = X\Y;               % calc the beta estimates
[3] u = Y-X*B_hat2;             % calc residuals
[4] s = sqrt( diag ((u'*u)/(length(Y)-4)*pinv(X'*X) )); % calc of std errors
[5] [ B_hat1 B_hat2 s B_hat2./s ]
ans =
    0.086784    0.086784    0.035517    2.4434
    1.3938    1.3938    0.039386    35.388
    -0.612    -0.612    0.063933    -9.5725
    0.20254    0.20254    0.039305    5.1528
    .....................................................................
```

This shows that the parameter estimates of B_hat1 and B_hat2 are identical.[12] The third column of ans shows the standard deviation of the parameter estimates and the last column the t-stat. The latter shows that all parameter are significantly different from zero at any reasonable level of confidence. A metric often applied to analyse the goodness-of-fit of a regression is called R^2, and is defined as the ratio of the sum of squares of the fitted value to the sum of squares of the observed values, i.e:[13]

$$R^2 = \frac{\beta^T X^T X \beta}{Y^T Y}.$$

We will now redo the regression calculations using the principle of Maximum Likelihood estimation. For this purpose it is assumed that the residuals from the regression are normally distributed so that the density function of the normal distribution can be used, as illustrated above. Matlab(R) uses an iterative approach to solve such problems implemented through the functions $fmincon$ and $fminunc$. The first handles constrained problems, i.e. where the parameters are restricted to lie within certain bounds or where a certain relationship between the variables need to be fulfilled. The latter handles optimisation problems without any types of constraints. A schematic presentation of the steps that need to be undertaken using Maximum Likelihood is:

1) Generate a guess for the parameters to be estimated;

[12] Matlab(tm) has a built in function "\" called "left matrix divide" which can also be used to solve the OLS regression.

[13] This is called the uncertered R^2.

2) Calculate the predicted values based on the guessed parameter values, i.e. insert the parameter guesses in the equation and apply data;

3) Calculate prediction errors by subtracting 2) from the observed left-hand-side variable

4) Insert the errors in the log-likelihood function which yields the log-likelihood function value

After the initial guess in (1), the steps (1)-(4) are repeated by the optimisation module in Matlab(R), where (1) is generated by Matlab(R) in a way that leads to the optimisation of the log-likelihood function value. All that the user has to do is to (a) write a function that returns the errors in (3) and supply the initial guesses mentioned in (1). This process is shown below by the use of a Matlab(R) script. A script file is simply a consecutive list of Matlab(R) commands similar to the ones from the examples above. In the examples, the instructions to Matlab(R) were invoked in the command line, i.e. one instruction at a time. In a script a series of commands can be collected and executed automatically by Matlab(R). To start writing the script we first initialise the MatLab(R) editor:

```
[1] edit
```

This opens a Matlab(R) editor window. Save the empty script file in a folder on your chosen drive and make sure to include the path in the Matlab(R) path structure.[14] The following script demonstrates the likelihood principle applied to the linear regression.

```
[1] % --- Generating variables ---
[2] R = xlsread('C:\A_projects\A_book\Data\Data','Rates');
[3] r = R(2:end,1); % allocates the 3m rate to variable r
[4] rLag3 = r(1:end-3,:); rLag2 = r(2:end-2,:); rLag1 = r(3:end-1,:); % lags
[5] Y = r(4:end,:); % adjusting r
[6] X = [ ones(length(Y),1) rLag1 rLag2 rLag3 ];
[7] % --- Initial and auxiliary variables/parameters ---
[8] T = length(Y);
```

[14]This is done by clicking on the "File" and "Set path" menus in the Matlab(tm) command window. Add the chosen path by browsing for it and save the path afterwards.

```
[9] B_0 = [.10; 1.40; -0.70; 0.20; .50];
[10] % --- Loglikelihood function ---
[11] lnL = @(B_0) T/2*log(2*pi*B_0(5,1))+...
[12]           1/(2*B_0(5,1))*sum( (Y-X*B_0(1:4)).^2 );
[13] % --- setting up the optimisation inputs ---
[14] options_ = optimset('LevenbergMarquardt', 'on', 'LargeScale', 'off', 'Display',
'off');
[15] lb = [-100;-100;-100;-100;0.001];
[16] ub = [100; 100; 100; 100; 100];
[17] B_hat = fmincon( lnL, B_0, [], [], [], [], lb, ub, [], options_)
```

This is a bit more involved compared to the closed form OLS calculations, however, the benefit of doing so is that the likelihood principle is generally applicable to other situations where explicit solutions do not exist, as also mentioned above. Lines [1]-[6] simply regenerate the data for the variables included in the regression. Line [8] calculates the number of time-series observations in the data, and line [9] specifies the starting values for the parameters to be estimated. Note that the ordering of the parameters is as above. A fifth variable is included which accounts for the variance of the residuals. When doing maximum likelihood optimisation it is almost always the case that the error-term variance is estimated along with the other parameters. Lines [11]-[12] calculate the likelihood function value.[15] Note that minus the log likelihood is calculated. This is done because $fmincon$ is designed to find the parameters that *minimise* the function value. If the built-in optimiser was designed to find parameters that *maximise* the function value, we would not multiply the objective function by minus one. Also note that line[11] uses the so-called function handle "@" to define the likelihood function within the script. Alternatively an external function could be written to calculate the likelihood function as shown in Section 2.5.2. Line [14] specifies which optimiser principle to use and lines [15]-[16] specify the lower and upper bounds for the parameters to be estimated. The only parameter that needs to be bound in our example is the variance. The lower bound for this parameter is set equal to 0.001 because a variance cannot be negative, while the upper bound is not constraining. Finally, line [17] contains the call to Matlab(R)'s built-in optimiser function, which solves

[15] This way of writing inline functions only works from Matlab(tm) version 7 on onwards. Alternatively, an inline function has to be defined by the "inline(...)" command.

the regression model and puts the optimal parameter values in the variable: B_hat. Note that line [17] does not end with a semicolon. This means that the result is printed to the screen. Running the above script by writing "[script name]" and pressing enter in the Matlab(R) command line gives the following output:

```
.....................................................................
[1] EX_likeli_1
Optimization terminated:  magnitude of directional derivative in search
direction less than 2*options.TolFun and maximum constraint violation
is less than options.TolCon.
No active inequalities.
B_hat =
0.086791
1.3938
-0.61198
0.20252
0.17376
.....................................................................
```

Line [1] contains the command that executes the script. Here, we have called the script "EX_likeli_1". Evidently, the estimated parameters are identical to the ones obtained by the closed form OLS expressions derived above. The standard errors on the parameter estimates can be calculated similarly by:

```
.....................................................................
[1] s = sqrt(diag(B_hat(5,1)*inv(X'*X)))
s =
0.035403
0.039259
0.063727
0.039179
.....................................................................
```

If the sample size is sufficiently large, it can be shown that the maximum likelihood estimates are distributed as:

$$\widehat{\beta} \approx N\left(\beta, T^{-1}IN^{-1}\right).$$

This means that the standard errors on the parameter estimates can be calculated from the inverse of the IN matrix, which is also called the information matrix. Here T denotes the number of observations and β denotes the true

parameter values. It turns out that Matlab(R) can produce an output that helps in this connection. In fact, *fmincon* can be called with additional output arguments of which some can be used to approximate the information matrix. This is done in the following way:

```
[1] [B_hat1,Fval,Exitflag,Output,Lambda,Grad,Hess] = fmincon( lnL, B_0, [], [], [], [],
lb, ub, [], options_);
```

where the relevant output is $Hess$, being the Hessian matrix i.e. the matrix of second derivatives to the objective function evaluated at the optimum with respect to the estimated parameters. $Hess$ can be used to proxy the information matrix when dealing with a minimisation problem since an estimate for the information matrix can be constructed by:[16]

$$\widehat{IN} = T^{-1} Hess.$$

When using $\widehat{IN}$ for IN the standard errors of $\widehat{\beta}$ are given by:

```
[1] s_hess = sqrt(diag(inv(Hess)))
s_hess =
0.037156
0.038585
0.06508
0.040874
0.0098444
```

Minor discrepancies between s and s_hess are observed. These stem from approximation errors of $Hess$ to the information matrix and from the approximative nature of the distribution of $\widehat{\beta}$ as indicated above.

2.8 A brief introduction to simulations

Many problems in finance do not have explicit solutions. In these cases, a simulation based solution strategy often proves to be helpful. For example, most asset and liability models rely on simulation based solutions; simulations are also used in the calculation of advanced risk measures for market

[16] Minus the hessian should be used for maximisation problems.

and credit risk; pricing of complex options relies heavily on simulations; and even portfolio optimisation can benefit from simulation based solutions. Simulation models can also be used as a back testing tool.

Only a very simple simulation example will be shown in this section. The underlying principle of this example remains the same also in more involved applications.

Remember the parameter estimates from the OLS regression above:

Parameter	Estimate	Standard deviation
k	0.087	0.037
a	1.394	0.039
b	-0.612	0.065
c	0.203	0.041
σ^2	0.174	0.010

And, assume that the parameter estimates are uncorrelated, i.e. orthogonal to each other, and that they are normally distributed. This means that we can simulate the regression equation:

$$\widehat{r}_t = k + a * \widehat{r}_{t-1} + b * \widehat{r}_{t-2} + c * \widehat{r}_{t-3} + \widehat{u}_t,$$

using the estimated parameters and their standard errors. Hereby simulated sample paths for r_t can be generated taking into account parameter uncertainty as well as the uncertainty that stems from the innovations to the process as represented by the error term. Often in simulation applications only a new innovation term is generated; either on the basis of an assumed distribution, or it can be regenerated from the observed errors (a technique doing this is referred to as: bootstrapping). In both of these cases the parameter uncertainty is ignored. However, there is no particular reason why to ignore parameter uncertainty; in fact, it all depends on the problem that one sets out to solve.

The current example is fairly general as it allows for both parameter uncertainty as well as uncertainty stemming from the innovation term. In order to generate the simulation, the following steps need to be completed:

1) draw innovations from the appropriate distribution;

2) generate parameter-value draws, also from appropriate distributions;

3) insert the generated parameters into the equation to be simulated;

4) feed innovations through the equation.

We show below how this can be done in Matlab through a script called "EX_sim".

```
[1] % --- load data ---
[2] R = xlsread('C:\A_projects\A_book\Data\Data','Rates');
[3] r_org = R(:,1);
[4]% --- Inputs ---
[5] b = [ 0.087; 1.394; -0.612; 0.203; 0.174 ];
[6] s = [ 0.037; 0.039; 0.065; 0.041; 0.010 ];
[7] init_ = [ 0.64; 0.76; 0.97; -0.097];
[8] nObs = 622;
[9] nSims = 500;
[10] Res = zeros(nObs,nSims);
[11] bSim = zeros(5,nSims);
[12]% --- Calculations ---
[13] for ( j=1:nSims ) % loop over simulations
[14]     b_hat = inf.*ones(length(b),1);
[15]     while ( sum(b_hat(2:4,1))>=1)
[16]        b_hat = b + s.*randn(5,1);
[17]     end
[18]     Y = zeros(nObs,1);
[19]     Y(1:3,1) = init_(1:3,1);
[20]     u = 0 + sqrt(b_hat(5,1)).*randn(nObs,1);
[21]     u(4,1) = init_(4,1);
[22]     for ( k=4:nObs ); % loop over observations
[23]        Y(k,1) = b_hat(1,1) + b_hat(2,1)*Y(k-1,1) + ...
[24]                              b_hat(3,1)*Y(k-2,1) + ...
[25]                              b_hat(4,1)*Y(k-3,1) + u(k,1);
[26]     end
[27]     Res(:,j) = Y;
[28]     bSim(:,j) = b_hat;
[29] end
[30]% --- generate stats ---
[31] bSim_m = mean(bSim')';
[32] bSim_s = std(bSim')';
[33] disp('original and simulates parameters...')
```

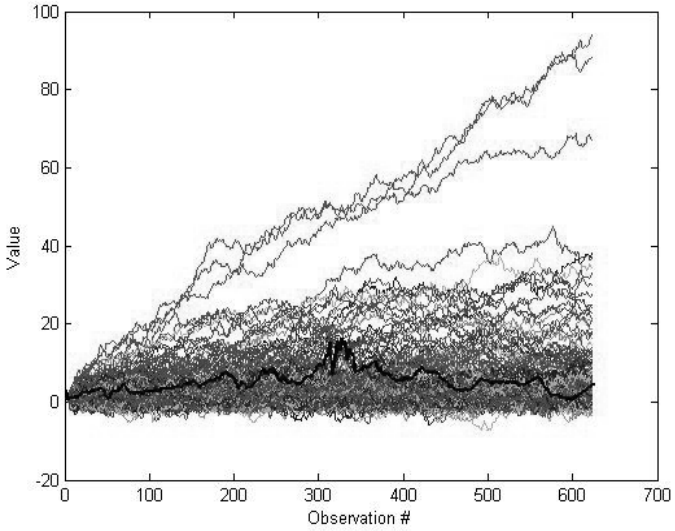
```
[34] disp([b bSim_m])
[35] disp('original and simulates standard errors...')
[36] disp([s bSim_s])
[37]% --- plot simulated and orginal series ---
[38] figure
[39] plot(Res)
[40] hold on
[41] plot(r_org,'k-','LineWidth',2)
[42]% --- plot distribution of simulated parameters ---
[43] figure
[44] x_1 = ( bSim_m(1,1)-4*bSim_s(1,1):bSim_s(1,1)/100:bSim_m(1,1)+4*bSim_s(1,1) )';
[45] pdf_1 = normpdf(x_1, bSim_m(1,1), bSim_s(1,1) );
[46] x_2 = ( bSim_m(2,1)-4*bSim_s(2,1):bSim_s(2,1)/100:bSim_m(2,1)+4*bSim_s(2,1) )';
[47] pdf_2 = normpdf(x_2, bSim_m(2,1), bSim_s(2,1) );
[48]% --- subplot 1
[49] [temp1 bin] = hist(bSim(1,:)',15);
[50] hist_obs = temp1'./nObs./(bin(1,2)-bin(1,1));
[51] subplot(1,2,1), bar(bin', hist_obs), title('Distribution for the constant');
[52] hold on
[53] plot(x_1,pdf_1);
[54]% --- subplot 2
[55] [temp1 bin] = hist(bSim(2,:)',15);
[56] hist_obs = temp1'./nObs./(bin(1,2)-bin(1,1));
[57] subplot(1,2,2), bar(bin', hist_obs), title('Distribution for 1st.  AR parameter');
[58] hold on
[59] plot(x_2,pdf_2);
```

The script generates two types of results. First it prints the original parameter values, and the mean of the simulated parameter values; then it plots the original data, simulated data trajectories and distributions for the first two simulated parameters (i.e. the constant and the first autoregressive parameter). This latter plot also contains a normal probability density function (pdf) fitted to the mean and standard deviations of these parameters.

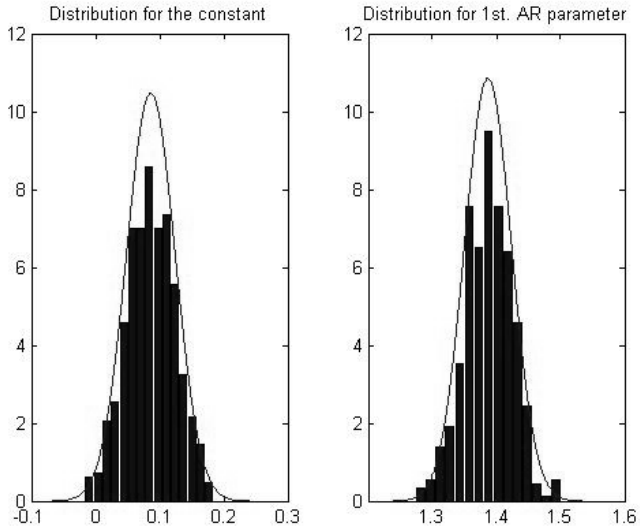
```
[1] EX_sim
original and simulates parameters...
0.087 0.081602
1.394 1.3845
```

```
-0.637 -0.65972
0.203 0.19231
0.174 0.1749
original and simulates standard errors...
 0.037 0.03789
 0.039 0.037638
 0.065 0.056403
 0.041 0.037679
 0.009 0.0088546
```

. . . . . .



Original and simulated data



Simulated parameter distributions.

The Matlab(R) program falls in three parts. The first part from lines [1]-[11] loads data and takes user input values for the number of simulations to be conducted and for the number of time-series observations to be generated in each simulation. The second part from lines [12]-[29] conducts the calculations. There is an outer for-loop covering the before mentioned lines which envelopes all calculations needed to generate one trajectory of the left-hand-side variable. In side this loop there is another for-loop that starts on line [22] and ends on line [26], which runs over the number of observations to be generated. It is within this for-loop that the AR(3) observations are generated. Finally, the third part of the program, covering the rest of the lines, generates summary statistics and reports results to the user in table and graphical format.[17]

2.8.1 Generating correlated random numbers

A central component of almost any experiment involving projections on the basis of an underlying model is that of correlated random numbers. Imagine for example that you want to simulate returns for a set of assets jointly. Naturally, the trajectories of these assets will be related: their particular co-

[17]The reader is invited to investigate the simulation code in more detail in the end-chapter exercises.

movements will be defined by the covariance matrix or the correlation matrix of these assets. A convenient way to generate correlated random numbers is by the use of the Cholesky decomposition of the covariance matrix. This decomposition allows for the calculation of the upper triangular matrix U from a positive definite[18] matrix C: a covariance matrix will always be positive definite. The Cholesky decomposition has the form (see, e.g. Glasserman (2004, pp.72-73)):

$$C = U^T U. \tag{2.2}$$

The upper triangular matrix U is useful for generating correlated random numbers in the following way:

$$R_{(r,c)} = n_{(r,c)} * U_{(c,c)}. \tag{2.3}$$

First uncorrelated random numbers can be generated of the desired dimension, e.g. $n \sim N(0,1)$. Then these random numbers are "run through" the Cholesky decomposition to generate the matrix R of correlated random numbers of the desired dimension, here (r, c).
An example illustrates this decomposition:

```
[1] disp('The covariance matrix')
[2] C = [ 7.9440 7.8265 8.3193 6.8492;
[3]       7.8265 7.7473 8.2614 6.8304;
[4]       8.3193 8.2614 8.8664 7.4661;
[5]       6.8492 6.8304 7.4661 7.1715 ]
[6] disp('Upper triangular from cholesky decomposition')
[7] U = chol(C)
[8] disp('Calculating transpose(U)*U')
[9] disp(U'*U)
[10] n = randn(500000,4);
[11] R = n*U;
[12] disp('Covariance matrix of simulated random variables')
[13] disp(cov(R))
```

Lines [2]-[5] define the covariance matrix to be used in the simulation; line [1] is just a print-to-screen command that gives a heading to what is printed

[18] The matrix $C_{[nxn]}$ is positive definite if $w^T C w > 0$, where $w_{[nx1]} \neq 0_{[nx1]}$.

next. Note that line [5] does not end with ";". This means that the matrix C is also printed to the screen. Lines [6]-[7] calculate and display the upper triangular matrix that is generated by the Cholesky decomposition. Lines [8]-[9] confirm the results of the decomposition by re-generating the covariance matrix following (2.2). Line [10] generates $500,000$ standard normal random draws and organises them in the matrix n of order $(nObs, nVars)$. Line [11] applies (2.3) to generate R, the matrix of correlated random numbers with the covariance structure given by C. Lines [12]-[13] calculate the covariance matrix of R to confirm that the calculations went well, and print the result to the screen. The following output is generated:

```
[1] The covariance matrix
[2] C =
[3]   7.9440 7.8265  8.3193 6.8492
[4]    7.8265 7.7473  8.2614 6.8304
[5]    8.3193 8.2614  8.8664 7.4661
[6]    6.8492  6.8304 7.4661  7.1715
[7] Upper triangular from cholesky decomposition
[8] U =
[9]    2.8185 2.77680 2.95170 2.43010
[10]   0      0.19121 0.34073 0.43149
[11]   0      0       0.19487 0.75075
[12]   0      0      0      0.71861
[13]Calculating transpose(U)*U
[14]   7.9440 7.8265 8.3193  6.8492
[15]   7.8265 7.7473 8.2614 6.8304
[16]   8.3193 8.2614 8.8664 7.4661
[17]   6.8492 6.8304  7.4661 7.1715
[18]Covariance matrix of simulated random variables
[19]   7.9243 7.8076  8.2998 6.8362
[20]   7.8076 7.7291  8.2427  6.8181
[21]   8.2998 8.2427  8.8474  7.4542
[22]   6.8362 6.8181 7.4542  7.1677
```

It can be seen that the covariance matrix calculated from the simulated data is marginally different from the starting covariance matrix, i.e. by comparing output lines [3]-[6] to lines [19]-[22]. The pattern of the simulated covariance matrix is however identical to the original covariance matrix, and this is what

is important. The minor differences originate from the randomness inherent in n. If fewer random draws than $500,000$ are included in n, the differences will tend to grow and if more than $500,000$ are included, they will tend to be smaller. It should be emphasised that there is nothing magical about the number $500,000$; this number is chosen here for illustrative purposes. In practise, one will often choose to simulate a number that is close to the number of observations contained in the original data sample used to estimate C in the first place.

The Cholesky decomposition together with a time-series model are the fundamental building blocks needed to generate forecasts/predictions.

Chapter 3

Fixed Income Preliminaries

3.1 Introduction

This chapter gives a very general introduction to fixed income markets. It presents the basic building blocks that we will draw on later on in the text such as yields and spotrates, and it shows how to calculate the price of bonds.

Learning objectives

- Understand the basic building blocks of fixed income markets
- Be able to work with yields and spotrates
- Know how to price bonds
- [to be completed]

3.2 Spot rates and yields

A first principle to get to terms with in finance is that: time has a monetary value. Not in itself, but in the sense that if one has the possibility to consume today rather than tomorrow, or at another point in the future, any person (also called an economic agent) has a preference to consume now. Consequently, agents in the economy have to be rewarded for postponing

consumption. If, in addition, the agents bear risks when postponing consumption i.e. there is uncertainty as to how much future consumption they receive, an additional compensation in the form of a risk premium applies. The reason why some agents accept to postpone consumption by e.g. depositing money in a bank or through the purchase of financial instruments, is that other agents in the economy are willing to pay the compensation rate (the interest rate) to obtain liquid funds (i.e. money). The economy is said to be in equilibrium, and a common interest rate emerges, when this latter group of agents (think of banks and firms that issue bonds and stocks to finance their investment projects) offer an interest rate at which their aggregate liquidity demand is fulfilled. The higher the interest rate offered, the more agents are willing to postpone consumption, and vice versa. The higher the interest rate requires, the lower the liquidity demand from firms, and vice versa. When liquidity demand and liquidity supply meet, the equilibrium is found, and a single interest rate clears the supply and demand for liquidity, at the particular segment of the yield curve - an argumentation similar to the above is often presented in the field of macro economics, where only one interest rate at a given maturity, often the very short end of the yield curve, is treated. In finance, however, we look at interest rates at different maturities and it is hence relevant to describe how rates for all maturities are generated. Several different hypothesis are put forth in this arena and they are loosely described below:

- The market segmentation hypothesis states that different agents have different preferences for borrowing or lending in different segments of the yield curve. For example, central bank investment operations may be most focused on the short term segment of the yield curve, investment banks on the medium term and pension funds on the longer term segments. The demand and supply within each segment determines the rate that is observed in that particular segment. If one segment is "under represented" in terms of either supply or demand, prices in this area will adjust as to attract agents from other segments. It thus implicitly assumed that agents, while having a preference for a particular segment, can be lured to invest in another segment if the compensation for switching segment is high enough.

- The expectation hypothesis states that todays' interest rates (spot rates) are determined on the basis of what future short rates are expected to be, and a risk premium; in other words: todays' yield curve

is the average of the future realisation of the short rate. In this way, the 2 year segment of todays' yield curve is the average of todays' short rate and the one year forward rate (plus a risk premium). This argumentation also works in reverse, hence if todays yield curve is upward sloping, it indicates that the future short rate is expected to increase (neglecting for a moment that adjustments have to be made for the risk premia).

- The liquidity preference hypothesis states that agents on average prefer to invest in short term assets, as this limits their exposure to interest rate risk. "Liquidity" is used as a term for how close the asset holding are to be liquid, in the sense of cash, if the assets are held to maturity. Longer dated bonds can naturally be more "liquid" than short dated bonds, in the sense of the word "liquid" which is usually used in finance, i.e. how fast an asset can be sold in the market place without having an impact on the prevailing market quotes. In the former sense of the word "liquid", this hypothesis assumes that agents can be convinced to invest in longer dated securities if they are compensated through a risk premium so that longer rates are higher than shorter rates.

The interest rate comes in different forms. For example, the real interest rate is the compensation that offsets the postponement of consumption to the future, without taking into account that goods tend to increase in price when time passes (i.e. the inflation rate). The nominal interest rate includes the real rate and the inflation rate. Interest rates are typically not identical for different time-horizons (also called maturities). Often the interest rate curve (also referred to as the yield curve) is upward sloping, reflecting that agents require a higher compensation rate the longer the consumption is postponed into the future; however, after a certain point in time agents seem to stop worrying about the time dimension of the consumption postponement, and the yield curve hence converges to a certain level. It is almost like the agents say, well it doesn't really matter whether I have to postpone my consumption ten or thirty years - this is the same to me, so I don't require extra a lot of extra compensation after year ten.[1] The reason for this is that agents react to the discounted value of the future consumption, and the marginal

[1]Here, ten is naturally just an example and the actual number varies over time and with the level of interest rates.

contribution is very small of consumption bundles that fall in the distant future.[2]

Figures 3.1 and 3.2 illustrate the time/maturity dimension of interest rates / yield curves. Figure 3.1 shows the US government yield curve at three different historical dates, and illustrates some of the generic forms that yield curves take: steeply upward sloping (May 2004); Inverse (March 1980); and normally upward sloping (February 1997). Figure 3.2 shows how the US yield curve has evolved over time since 1953.

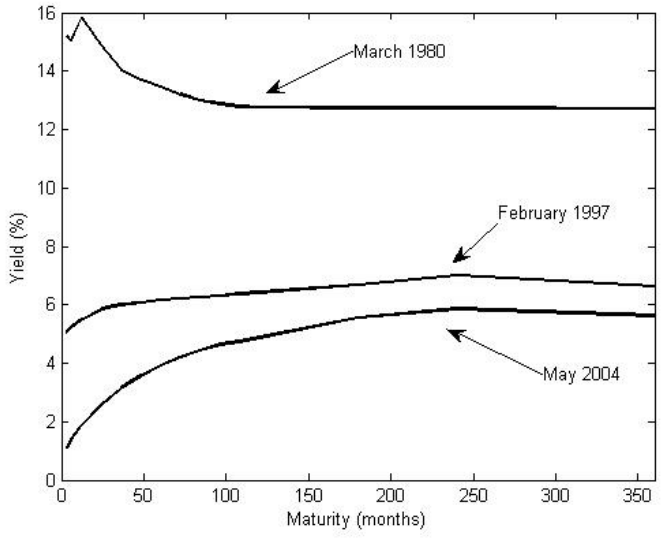


Figure 3.1: Yield curve examples on given dates.

Consumption bundles are not directly observed in the financial markets. However prices of e.g. bonds are, and these prices will hence reflect the interest rate required by the bond holders. In effect, bond prices can be used to calculate interest rates. The more actively traded bonds will have more precisely determined prices (because more agents evaluate them before trading them) and will thus better reflect the implied interest rate than the less traded bonds will. It is therefore important to use highly traded/liquid bonds in the calculation of the market interest rate curve.

[2]Net present value calculations can confirm this point by the fact that the discounting function is convex and negatively sloped.

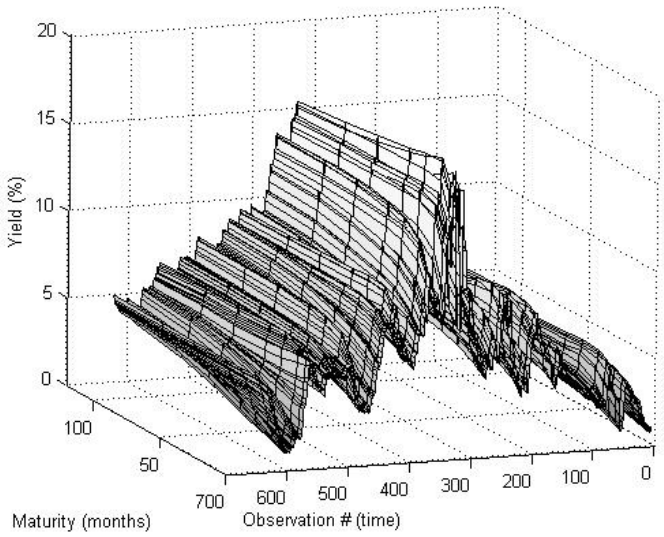


Figure 3.2: Example of a yield curve shapes and locations

A fixed rate bond is like a loan: the bond seller promises the buyer to pay fixed coupons over time, until the bond matures, and when it does the seller will repay the principal amount borrowed. Bonds can be issued at different loan-sizes (notional amounts).[3] For the purpose of this text we assume that all bonds are issued with a notional amount of 100 (insert your favorite currency, e.g. Euro) and have annual payments. Before maturity the price of a bond is determined by its cashflow and the discount factors:

$$P_t = \frac{C}{(1+r_1)^1} + \frac{C}{(1+r_2)^2} + \cdots + \frac{C+100}{(1+r_n)^n} = C^T * D, \qquad (3.1)$$

[3]Other types of bonds also exist such as floting rate bonds, where the promissed coupon payments adjust to the current level of rates over time; step-up-bonds where the coupon is corrected if the credit rating of the bond changes over time; lottery bonds, where the maturity time depends on a lottery (usually it is the state that offers these bonds) and the redemption value can be higher than the face value of the bond; zero coupon bonds that have no intermediate coupons over the maturity of the bond; perpetual bonds that have no maturity date; and convertible bonds that can either allow for early redemption by the holder of the bond, or if issued by a company such bonds may include an imbedded right to convert the bond into equity. These type of bonds are not treated in the text, only fixed rate bonds are.

where $C_{(n,1)}$ collects the cash flows (i.e. the coupons) and $D_{(n,1)}$ the discount factors, i.e. the terms $(1 + r_j)^{-j}$ for $j = \{1, 2, \ldots, n\}$. The present value of each intermediate coupon payments and the final payment at time-points $\{1, 2, \ldots, n\}$ is calculated as the product of the payments and the discount factor. The discount factor is the inverse of one plus the interest rate to the power of the time-period. To illustrate how the relation in (3.1) can be used to find the interest rates implied by the bond prices consider the following example. Let's assume that five actively traded bonds and one less actively traded bond form our particular investment universe, and that the following are observed:

Bond	Price	Coupon (annual)	Maturity (years)
A	100.00	5.0%	1
B	99.50	5.5%	2
C	93.75	4.0%	3
D	89.45	3.5%	4
E	91.00	4.5%	5
F	?	3.0%	3

In the light of (3.1) this information can be collected in the following way:

$$P_{(n,1)} = C_{(n,n)} * D_{(n,1)} \Longleftrightarrow D_{(n,1)} = C^{-1}_{(n,n)} * P_{(n,1)}. \tag{3.2}$$

Remembering that D is the vector of discount factors, the interest rates are defined as:

$$r_j = \left(D_j^{-1/j}\right) - 1 \quad \forall j = \{1, 2, \ldots n\}.$$

In Matlab(R) this can be implemented in the following way:

```
[1] j = (1:1:5)';
[2] P = [100.00; 99.50; 93.75; 89.45; 91];
[3] CF = [105.0 0.0 0.0 0.0 0.0;
[4]    5.5 105.5 0.0 0.0 0.0;
[5]    4.0 4.0 104.0 0.0 0.0;
[6]    3.5 3.5 3.5 103.5 0.0;
[7]    4.5 4.5 4.5 4.5 104.5 ];
[8] D = inv(CF)*P;
```

```
[9] R = D.^(-1./j)-1
```

Note that line [9] uses element-by-element calculations by ".^" and "./". Executing this script gives the following result:

$$R = [0.050000,\ 0.057933,\ 0.063888,\ 0.066228,\ 0.067242]^T .$$

R is the vector of spot rates and this contains the annual rates for $1, 2, 3, 4,$ and 5 years investments, i.e. the compensation rate that agents require for investing at these horizons.

It is now possible to find a fair price for bond F using the information about F's cash flows and the spot rates in R. This is done in the following way:

```
[1] Pf = [3 3 103]*(1+R(1:3)).^(-j(1:3,1))
    Pf =
     91.074
```

The other yield curve concept mentioned above: "yield to maturity" is a complicated average of the spot rates contained in R. The yield to maturity is the same as the internal rate of return on the stream of cash flows that define the bonds in the investment universe, and can thus only be calculated once the bond prices are known.

Yield to maturity is calculated in the following way:

$$P_t = \frac{C}{(1+y)^1} + \frac{C}{(1+y)^2} + \cdots + \frac{C+100}{(1+y)^n}. \tag{3.3}$$

An expression like (3.3) cannot be solved directly because the variable of interest (y) enters in the expression as a polynomial of order n. Luckily, Matlab(R) can find a solution using numerical techniques.[4]

```
[1] function [out] = price2yield(in)
[2]% Chapter:  Risk and Return
```

[4]The Financial Toolbox of Matlab(R) contains a built-in function called *irr* that can handle these calculations. However, in case the Financial Toolbox is not available the example in the text demonstrates how the internal rate of return (yield) can be calculated. The built-in Matlab(tm) function relies on the root-finding function *root*, whereas the example in the text draws on *fmincon*.

```
[3]%
[4]% Usage:
[5]% [out] = price2yield(in)
[6]%
[7]% in (structure):
[8]% .P - is the price of the cashflow 1-by-1
[10]% .CF - cashflow [-price, cpn1, cpn2,...,cpn(r)+principal] r-by-1
[11]% .t - timing of cashflows in years [0;1;2;...;r] r-by-1
[12]% e.g.  6 months should be written as 0.5
[13]% out (structure):
[14]% .Y - vector of yields
[15]%
[16]% date:  October 2006
[17]% report bugs to:  email@kennyholm.com
[18]%
[19] warning off all; clear y;
[20] P = in.P;
[21] CF = in.CF(:);
[22] t = in.t(:);
[23] nObs = length(CF);
[24]% --- Defining the function ---
[25] Fx = @(d)(-P+CF'*(d.*ones(nObs,1)).^(t))^2;
[26]% --- Finding the optimum
[27] options_=optimset('LevenbergMarquardt','on','LargeScale','off','Display','off');
[28] [D] = fmincon(Fx, 1, [], [], [], [], 0, 2, [], options_);
[29] out.Y = 1/D-1;
```

This function, called *price*2*yield*, calculates the yields for individual bonds as they are defined by their cash flows. Line [1] defines the function name and the input and output arguments. It should be noted that structured variables are used for the inputs and output. For the output this could be seen as a slight over–kill since only one output is generated, however, it is sometimes easier to stick with a more general approach than to accommodate individual function needs. In the end, this is a matter of taste. Lines [2]-[18] contain user help information on which format should be used when supplying the function with input variables, as well as information on how to execute the function. Lines [19]-[23] take care of some general settings and parameter definitions: warnings are turned off to ensure that the user does

not get messages intended for the developer of the function; the variable to be calculated y is cleared to ensure that old values are not carried over to the current calculation; the user supplied price of the bond is allocated to the variable P; the cash flow supplied by the user is allocated to the variable CF; and the time-steps to the variable t; and the number of observations in stored in the variable $nObs$. Line [25] contains the function to be minimised: it is defined as the squared residuals from (3.3), i.e.

$$\min_D F(D) = \left[-P_t + C * D^1 + C * D^2 + \cdots + (C + 100) * D^n\right]^2.$$

Note that a function handle is used to define the objective function in line [25]. This handle is then "called" in line [28], which draws on the built-in optimiser $fmincon$ to conduct the minimisation. D is returned as the optimal value for the discount factor i.e. $D = 1/(1 + r)$. Consequently, line [29] calculates the yield y from the discount factor and passes this value to the function's output structure.

The following script calculates yields from the example bond-universe given above.

```
[1] j = (1:1:5)';
[2] P = [100.00; 99.50; 93.75; 89.45; 91];
[3] CF = [105.0 0.0 0.0 0.0 0.0;
[4]           5.5 105.5 0.0 0.0 0.0;
[5]           4.0 4.0 104.0 0.0 0.0;
[6]           3.5 3.5 3.5 103.5 0.0;
[7]           4.5 4.5 4.5 4.5 104.5 ];
[8] for ( j=1:5 )
[9]    in.P = P(j,1);
[10]    in.CF = CF(j,1:j);
[11]   in.t = (1:1:j)';
[12]   [out] = price2yield(in);
[13]   yield(j,1)=out.Y;
[14] end
[15] yield'
```

Lines [1]-[7] allocate the input to variables. The for loop in line [8]-[14] performs the calculations for each bond's cashflow and stores the yield in

the vector: *yield*. Line [15] prints the yields for years 1-5, and the output is shown in line [16].

It is now interesting to compare the one period rates (also called zero coupon rates or spot rates) calculated following (3.1) to the yields calculated according to (3.3). Figure 3.3 draws the comparison.

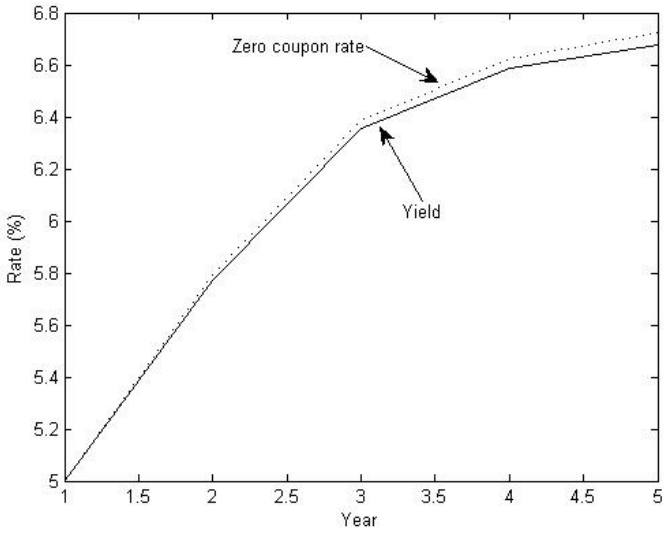


Figure 3.3: Comparison of a zero curve and a yield curve

It can be seen that the yield curve lies below the zero coupon curve, for all years but the first. This is not a coincidence. It can be seen from the definitions of the yield in (3.3) and the zero coupon curve in (3.1) that the two calculations are identical for the rate applying to the first time-period. After this the first time-period, the yield can be viewed as a weighted average of the spot rates, where the coupon payments are the weights. Hence, if the structure of one-period rates is increasing in maturity so that $r_0 < r_1 < r_2 \ldots$ the curve of yields will be below the zero coupon curve; conversely, if the structure of rates is downward sloping, i.e that $r_0 > r_1 > r_2 \ldots$ the zero coupon curve will be below the curve of yields. If the structure is flat the zero coupon curve and the curve of yields will be identical.

3.3 Forward rates

A concept that is intimately related to spot rates is that of forward rates. This concept covers the transaction where money is borrowed or lend between two future dates on terms that are agreed upon today. When assuming that the actions of financial market participants lead to efficiently determined bond prices, it is evident that spot rates and forward rates are inseparable so that forward rates can be uniquely inferred from spot rates and vice versa. Consider the spot rates determined above for year zero to one and from year zero to two, i.e. r_1 and r_2, and the forward rate from year 1 to year 2, called $f_{1,2}$. There are two strategies available to the investor who wants to invest over a two year horizon: (a) invest in a two year bond earning $(1 + r_2)^2$, or (b) investing in the one year bond for one year and in the forward contract from year one to year two. If financial markets are effective these two strategies will bring the investor identical monetary compensations because both (a) and (b) can be traded at time 0. Effectively, the following relationship must hold between spot and forward rates:

$$(1 + r_2)^2 = (1 + r_1) * (1 + f_{1,2}),$$

which means that the forward rate can be determined by rearranging terms:

$$f_{1,2} = \frac{(1 + r_2)^2}{1 + r_1} - 1.$$

This formula can be generalised for annual rates as:

$$f_{n,m} = \left[\frac{(1 + r_m)^m}{(1 + r_n)^n}\right]^{1/(m-n)} - 1 \quad for \;\; m > n. \tag{3.4}$$

Using the spot rates depicted in Figure 3.3 and (3.4) the corresponding one-period forward rates can be determined. This is done in the example below, where R contains the spot rates and F contains the forward rates:

```
k = (1:1:5)';
P = [100.00; 99.50; 93.75; 89.45; 91];
CF = [105.0 0.0 0.0 0.0 0.0;
     5.5 105.5 0.0 0.0 0.0;
     4.0 4.0 104.0 0.0 0.0;
     3.5 3.5 3.5 103.5 0.0;
```

```
     4.5 4.5 4.5 4.5 104.5 ];
%...  Calculating spot rates
D = inv(CF)*P;
R = D.^(-1./k)-1;
%...  Calculating Forwards
F = ((1+R(2:end,1)).^k(2:end,1)./(1+R(1:end-1,1)).^k(1:end-1,1))-1;
F = [R(1,1);F];
.....................................................................
```

3.4 Bond pricing functions

Once the yield of a coupon paying bond is known, it is possible to write the bond pricing function in a short-hand way, rather than using the summing formula in (3.5):

$$P_t = \sum_{j=1}^{n} \frac{C}{(1+y)^j} + \frac{100}{(1+y)^n}. \tag{3.5}$$

To obtain this short-hand pricing function a tool is needed that allows for re-writing of (3.5). Such a tool is obtained in the following way[5]: let $g = \sum_{j=k}^{m} q^j$ and multiply both sides of this expression by q. This gives $q * g = \sum_{j=k+1}^{m+1} q^j$. Then calculate $g - q * g = \sum_{j=k}^{m} q^j - \sum_{j=k+1}^{m+1} q^j$, which is the same as $(1-q) * g = q^k - q^{m+1}$, where the LHS follows directly after taking g outside brackets, and the RHS follows because all terms overlap in the two sums that are subtracted, apart from the terms involving q^k and q^{m+1}. Now, dividing both sides by $(1-q)$ which gives:

$$\sum_{j=k}^{m} q^j = \frac{q^k - q^{m+1}}{1-q}.$$

Setting $q = 1/(1+y)$, this expression can be used to re-write (3.5) to:

[5] See also Tuckman (2002, p.42).

$$
\begin{aligned}
P_t &= C * \sum_{j=1}^{n} \frac{1}{(1+y)^j} + \frac{100}{(1+y)^n} \\
&= C * \left[\frac{1/(1+y) - 1/(1+y)^{n+1}}{1 - 1/(1+y)} \right] + \frac{100}{(1+y)^n} \\
&= C * \left[\frac{\frac{1}{1+y} - \frac{1}{(1+y)^{n+1}}}{\frac{1+y}{1+y} - \frac{1}{1+y}} \right] + \frac{100}{(1+y)^n} \\
&= C * \left[\frac{\frac{1}{1+y} - \frac{1}{(1+y)^{n+1}}}{\frac{y}{1+y}} \right] + \frac{100}{(1+y)^n} \\
&= C \left[\frac{1}{y} - \frac{1}{y} * \frac{1}{(1+y)^n} \right] + \frac{100}{(1+y)^n} \\
&= \frac{C}{y} \left[1 - \frac{1}{(1+y)^n} \right] + \frac{100}{(1+y)^n}. \qquad (3.6)
\end{aligned}
$$

One advantage of (3.6) over (3.5) and (3.1) for that matter, is that it presents a closed form expression for the price in the sense that it does not rely on a sum. When implemented in Matlab(R), or any other programming language, the calculations of a simple expression will be faster and easier, compared to the looping structure that is necessary to handle the calculation of a sum.

[TBI: derive an expression similar to the above in continous time...]

3.5 Exercises

Exercise 1 *Replicate Figure 4.4 using the bond function as shown in the text.*

Chapter 4

Risk and Return Measures

4.1 Introduction

This chapter presents the principles for calculating risk summary statistics for fixed income securities. These are essential building blocks that help to extract information relevant for doing forward looking investment decisions.

Learning objectives

- Understand the basic risk concepts relevant for fixed income instruments
- Be able to calculate returns for fixed income instruments
- Understand which factors that drive yield curve changes

4.2 Risk Measures

A key element in financial management and investment operations is proper handling of risks. In particular, to facilitate optimal decision making, it is necessary to estimate the risk of individual assets and portfolios in a comprehensive way. Traditional investment analysis, e.g. the Capital Asset Pricing Model, puts emphasis on variance/standard deviation as a risk measure. The variance if defined as:

$$\sigma^2 = E\left[(y-\overline{y})^2\right] = \frac{1}{J-1}\sum_{j=1}^{J}(y_j-\overline{y})^2\,,$$

where y is the observed quantity, e.g. a time-series of returns, and $\overline{y}$ indicates the average. The standard deviation is defined as the square root of the variance, i.e.

$$\sigma = \sqrt{E\left[(y-\overline{y})^2\right]}.$$

In addition to the variance, the covariance is an important measure, which is defined as:

$$\sigma_{j,k} = E\left[(x-\overline{x})(y-\overline{y})\right] = \frac{1}{J-1}\sum_{j=1}^{J}(x_j-\overline{x})(y_j-\overline{y}) = \sigma_j * \sigma_k * \rho_{j,k}$$

where ρ is the correlation coefficient. The variance and covariance is collectively called the second moment of a distribution and communicates how dispersed the series observations are around the average value and how the data series move together over time. To illustrate this concept assume that the following information is given about x and y:

$$\left[\begin{array}{c} x \\ y \end{array}\right] \sim N(\mu, \Omega)\,,$$

which means that x and y are jointly normally distributed with mean μ and covariance matrix Ω. The covariance matrix has the variances on the diagonal and covariances as the off-diagonal elements. Assume further that

$$\mu = \left[\begin{array}{c} 0 \\ 0 \end{array}\right], \quad \text{and} \quad \Omega = \left[\begin{array}{cc} 1.00 & 1.2 \\ 1.2 & 4.00 \end{array}\right].$$

This implies that the correlation between x and y is $\rho_{x,y} = \sigma_{x,y}/\sigma_x\sigma_y = 0.6$. Graphically this information is represented in Figure 4.1. The upper panel shows the two marginal distributions; the distribution for y has higher variance than the distribution for x. The lower panel shows a plot of the pairwise observations for x and y. It is clear that there exist a positive correlation/covariance between the variables: the elliptic shape of the plotted area shows that when x is high then y tends to be high, and when x is low y tends

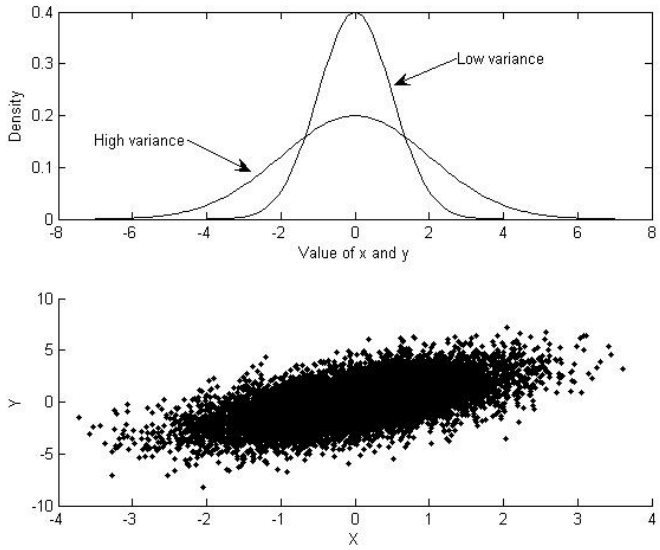


Figure 4.1: Variance and covariance

to be low. If there was no linear relation between x and y then the plot would be circular. The variance/standard deviation and covariance are comprehensive risk measures if the underlying variables are normally distributed; and even when the analysed variables are non-normal, which is the case for most financial variables, will the covariance often be an informative measure.

4.2.1 Value-at-risk and Expected Shortfall

The risk measures Value-at-Risk (VaR)[1] and Expected Shortfall (ES) will often supplement and sometimes substitute the standard deviation as a risk measure. Considering a return distribution for x, the VaR figure is the largest possible (often negative) return, at a given confidence level and over a given period of time, that one will experience; the ES is the expected shortfall beyond the VaR level. The following definitions apply for the return distribution of x:

[1]The Value-at-Risk (VaR) should not be confused with VAR, which is an acronym for the econometric model: Vector Auto Regression.

$$VaR_\alpha(x) = \max\{l : \Pr[x \le l] \le 1-\alpha\} = F_x^{-1}(1-\alpha) \quad (4.1)$$
$$ES_\alpha = E[x \,|\, x \le VaR_\alpha(x)]. \quad (4.2)$$

α is called the confidence level, so for for example $\alpha = 0.99$, refers to a 99% confidence level. F^{-1} refers to the inverse of the cumulative return distribution of x. The intention of (4.1) is to express VaR as the left tail in the return distribution (which will most likely be a negative number), so that there is no more than $(1-\alpha)$ probability mass in the tail beyond l. This is equivalent to the inverse of the cumulative distribution function of the returns x at the $(1-\alpha)$ confidence level. Equation (4.2) defines the expected shortfall as the average return beyond the VaR level. Figure 4.2 illustrates these risk measures on the basis of a probability density function of returns / portfolio values.

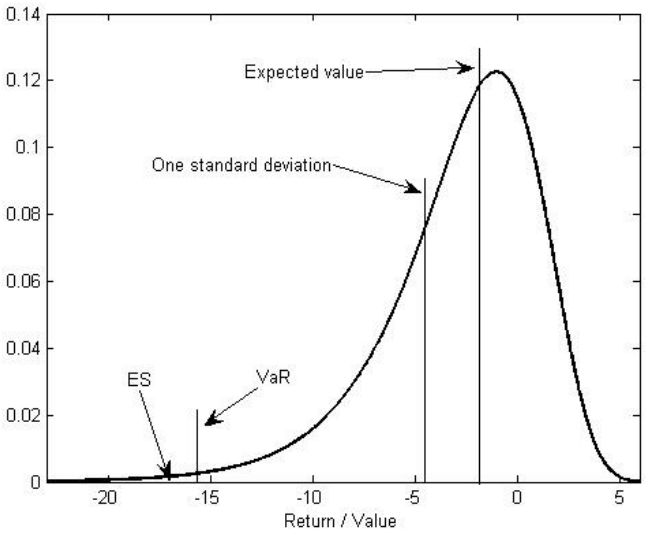


Figure 4.2: Risk measures

Sometimes VaR is defined on the distribution of losses, L . In this case Figure 4.2 is reversed so that the tail of interest is the one to the right in the picture. In this case VaR can be defined as:

$$VaR_{\alpha}(L) = \min\{l : \Pr[L \geq l] \leq 1 - \alpha\}$$

In the example below these risk concepts are calculated using the Dow Jones equity index contained in the Excel file: Data.xls.

```
[1]% --- load and sort data ---
[2] Dat = xlsread('C:\A_projects\A_book\Data\Data','Equity');
[3] stepp = 0.5;
[4] [nObs junk] = size(Dat);
[5]% --- calc summary stats ---
[6] Ret_s = sort(Dat(:,4));
[7] avg = mean(Ret_s); sig = std(Ret_s);
[8] nObs_99 = floor(.01*nObs); nObs_95 = floor(.05*nObs);
[9] VaR_99 = Ret_s(nObs_99,1); ES_99 = mean(Ret_s(1:nObs_99-1,1));
[10] VaR_95 = Ret_s(nObs_95,1); ES_95 = mean(Ret_s(1:nObs_95-1,1));
[11] VaR_95_N = avg-norminv(0.95,0,1)*sig;
[12] VaR_99_N = avg-norminv(0.99,0,1)*sig;
[13]% --- generate normal pdf ---
[14] x = (VaR_99:stepp:max(Ret_s))';
[15] ret_norm = pdf('norm',x,avg,sig);
[16]% --- generate bar plot of observed returns ---
[17] [bar1 bin] = hist(Ret_s,25);
[18] bar1_adj = bar1'./nObs./(bin(1,2)-bin(1,1));
[19] figure
[20] bar(bin', bar1_adj,'w')
[21] hold on
[22] plot(x,ret_norm,'k-','linewidth',2)
[23]% --- generate output table ---
[24] disp('...Results...')
[25] disp(sprintf('VaR(99)= %6.2f Based on Normal Dist:  %6.2f ', [VaR_99 VaR_99_N]))
[26] disp(sprintf('VaR(95)= %6.2f Based on Normal Dist:  %6.2f ', [VaR_95 VaR_95_N]))
[27] disp(sprintf('ES(99) = %6.2f ES(95) = %6.2f ', [ES_99 ES_95]))
```

This script generates the following output on VaR and ES as well as Figure 4.3. The Figure shows the empirical return distribution overlaid with a fitted normal distribution.

```
[1]...Results...
[2]VaR(99)= -11.00  Based on Normal Dist:  -9.25
[3]VaR(95)= -6.44   Based on Normal Dist:  -6.37
[4]ES(99) = -16.58  ES(95) = -9.47
```

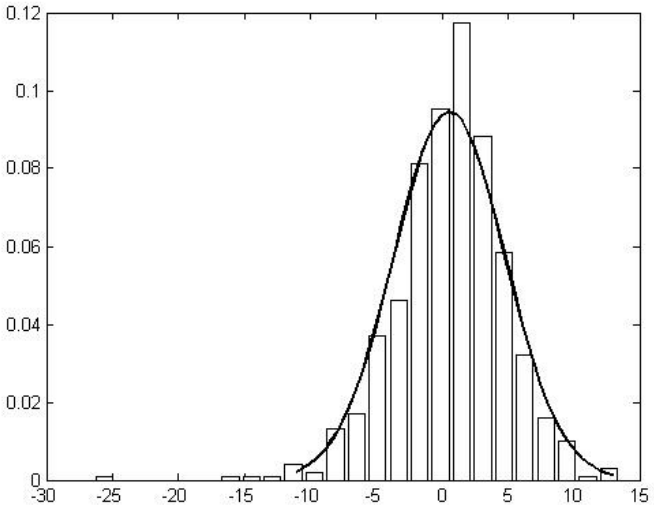


Figure 4.3: Empirical and normal distribution

In the above script, lines [1]-[4] load the data, define a step-length parameter and find the number of observations of the loaded data. Lines [5]-[12] perform the central calculations of the script file. Line [6] sorts the return series from smallest value to largest value; line [7] calculates the mean and the standard deviation of the returns; line [8] finds the observation number that corresponds to the 1% and 5% tail-point of the return distribution. The actual return observation at these observation numbers are the empirical VaRs at the 95% and 99% confidence levels. The *floor* function is used to round the observation number to an integer value, since the observation number has to be an integer. Using *floor* rather than *ceil* produces a more conservative estimate of the VaR number. For the case at hand: there are 624 return observations, which implies that the 99% VaR corresponds to

observation numbers $floor(0.01 * 624) = 6$ or $ceil(0.01 * 624) = 7$. Similar calculations can be performed for the 95% VaR giving observation numbers 31 and 32, respectively. Line [9] is a "lookup" in the sorted vector of returns called *Ret_s* for the 99% VaR. The second part of Line [9] calculates the Expected Shortfall i.e. the average of the observations that are above the VaR level. Line [10] performs similar operations for the 95% level. Lines [11] and [12] calculate the VaR based on a normal distribution using the following formula:

$$VaR_{\alpha} = E[r] + \Phi^{-1}(1-\alpha) * \sigma * \sqrt{t}, \tag{4.3}$$

where Φ^{-1} is the inverse normal distribution. The $\sigma\sqrt{t}$ in (4.3) accounts for the standard deviation of the return series over the period of time that the VaR number is calculated for. In the normal distribution the variance scales with time, so the standard deviation scales by the square root of time. For example, if the standard deviation is calculated as the per annum value and a one month VaR is required, then $t = 1/12$. In the calculation performed in the script file above, the VaR is calculated for the observation frequency of the return observations, hence no time-adjustment is necessary.

Expected shortfall in closed form

A closed form expression for the expected shortfall can be derived when assuming that the returns are normally distributed. Starting from the definition of the expected shortfall[2]

$$ES_c(x) \equiv \frac{1}{1-c}\int_0^{1-c} Q_x(s)\ ds \equiv \frac{1}{1-c}\int_0^{1-c} F_x^{-1}(s)\ ds, \tag{4.4}$$

where Q is the quantile and F is the cumulative distribution function. The variable c is the confidence interval e.g. c=99% implies that the expected shortfall is calculated for the 1% worst outcomes of the variable under investigation. Naturally, the definition in (4.4) is identical to (4.2). Assuming that $x \tilde{} N(0,1)$, the closed from expression for the expected shortfall is given by:

$$ES_c(x) = \frac{-1}{(1-c)\sqrt{2\pi}} \exp\left\{-\left[\frac{VaR(1-c)^2}{2}\right]\right\}. \tag{4.5}$$

[2]See, e.g. **?**

To derive (4.5) the following building blocks are needed:[3]

$$F^{-1}(s) = \sqrt{2}\,\mathrm{erf}^{-1}(2*s-1) \tag{4.6}$$

$$\int_0^w \mathrm{erf}^{-1}(s)\ ds = \frac{-1}{\sqrt{\pi}}\exp\left\{-\left[\mathrm{erf}^{-1}(s)\right]^2\right\}. \tag{4.7}$$

Substituting (4.6) into (4.4) gives:

$$\begin{aligned} ES_c &= \frac{-\sqrt{2}}{1-c}\int_0^{1-c}\mathrm{erf}^{-1}(2*s-1)\ ds \\ &= \frac{-\sqrt{2}}{(1-c)*2*\sqrt{\pi}}\exp\left\{-\left[\mathrm{erf}^{-1}(1-2*c)\right]^2\right\} \\ &= \frac{-1}{(1-c)\sqrt{2\pi}}\exp\left\{-\left[\mathrm{erf}^{-1}(1-2*c)\right]^2\right\}, \end{aligned} \tag{4.8}$$

where the second line follows from the integration rule: $\int f(as+b)\,ds = \frac{1}{a}F(as+b)$ in combination with (4.7).[4] Line 3 simplifies the expression. Because of symmetry of $F(s)$ and $\mathrm{erf}^{-1}(s)$, $s \in \{0,\ldots,1\}$, and using (4.6) we can write:

$$\begin{aligned} &\left[\sqrt{2}\,\mathrm{erf}^{-1}(1-2*c)\right]^2 = \left[F^{-1}(1-c)\right]^2 \\ &\Updownarrow \\ &\left[\mathrm{erf}^{-1}(1-2*c)\right]^2 = \frac{\left[F^{-1}(1-c)\right]^2}{2} = \frac{VaR(1-c)^2}{2}, \end{aligned}$$

which equates (4.8) and (4.5).

To validate the derived expression, the below Matlab(R) script compares the application of (4.5) to expected shortfall values calculated on the basis of simulations.

```
[1] CI = [ 0.90; 0.95; 0.975; 0.99; 0.999; 0.9999 ];
```

[3] See the appendix for a description of the error function (erf).

[4] The character F denotes the primitive function of f such that $F'(x) = f(x)$. It is important to distiguish between F and F, where the latter is the cumulative distribution function.

```
[2] x = randn(1e6,1);
[3] nCI = length(CI);
[4] ES_sim = zeros(nCI,1);
[5] ES_calc = zeros(nCI,1);
[6] for ( j=1:nCI )
[7]    c = CI(j,1);
[8]    ES_sim(j,1) = sum((x<=norminv(1-c)).*x)/sum(X<=norminv(1-c));
[9]    ES_calc(j,1) = -1/((1-c)*(2*pi)^0.5)*exp(-(norminv(c)^2)/2);
[10] end
[11] disp('Simulated and calculated numbers')
[12] disp(' CI ES_sim ES_calc ')
[13] disp([CI ES_sim ES_calc])
```

Line [1] defines the confidence levels at which the calculations are to be performed. In line [2] return realisations are generated, as one million draws from the standard normal distribution. These draws are stored in a variable called x, which forms the basis for the calculation of the simulation-estimate of the expected shortfall. Line [3] finds the number of confidence levels for which calculations are performed. This line is not strictly needed since we should be able to remember how many confidence levels we entered in line [1]. However, requesting the length (i.e. the number of entries) of the variable CI makes the script more flexible and easier to adapt if we want to run the experiment for a higher or lower number of confidence levels. In this case it is necessary only to make changes to the script in line [1]. Otherwise, we would need to change lines [1],[3]-[6]. Lines [4]-[5] clear the vectors that store the estimated numbers that are generated by the loop in lines [6]-[10]. Line[7] allocated a given confidence level to the scalar c taken from the vector of all relevant confidence levels, CI, on the basis of the looping variable j. j spans the integers from 1 to nCI, as seen in line [6]. In the example above $nCI = 6$. Line [8] generates the simulation based estimate of the expected shortfall. This is done by first finding the realisations of x that are below the threshold as specified by the VaR of the respective confidence interval. In Matlab(R) the relation operators can be used on vectors to generate indicator variables, in effect the expression $(z <= Z)$, generates a vector of 0s and 1s, of the same length as z and Z, where a value of 1 represents when the "test", in our case "less then or equal to", is true and a value of 0 represents that the test is false. It can then be seen that line [8] calculated an indicator variable for all outcomes of x that fall in the left tail of the distribution, where the left

tail is defined by $VaR(1-c)$. The product of the indicator variable and x, i.e. `(x<=norminv(1-c)).*x)`, is a vector of 0 and $x_i s$, where x_i represent a given ith element of x falling in the left tail. The sum of this vector approximates the integral of the tail and the denominator of line [8] contains the number of observations in the tail. In this way line [8] calculates the average of the occurances of x that fall in the left tail i.e. it calculates the simulation based expected shortfall. Line [9] implements (4.5). Lines [11]-[13] prints the results. Running the script generated the following output:

```
Simulated and calculated numbers
   CI       ES_sim    ES_calc
 0.9000    -1.755     -1.755
 0.9500    -2.065    -2.065
 0.9750    -2.340    -2.339
 0.9900    -2.669    -2.665
 0.9990    -3.366    -3.367
 0.9999    -3.947    -3.959
```

Evidently, the simulated and calculated numbers are close to each other. Any discrepancy between the numbers arise from approximation errors in the estimate derived from the simulation based expected shortfall calculation.

4.2.2 Duration and modified duration

One of the key risk measures in the fixed income investment universe is the modified duration. This measure has at least two interpretations, as:

a) a present value weighted average of future payments/cashflows[5]

b) the bond price sensitivity to a change in the underlying interest rates curve

It is the second of these interpretations that is most frequently applied in fixed income analysis and the one that we will focus on in this section. To illustrate the concept of modified duration it is instructive to examine the relationship between the price of a bond and its yield. For this purpose we

[5]This is actually the interpretation of "duration"; modified duration is the duration divided by one plus the yield.

run the script below that calculates the price of four bonds for varying levels of yields. The results are plotted in Figure 4.4.

```
[1] Bond1 = [ 3 3 3 3 103 ];
[2] Bond2 = [ 3 3 3 3 3 3 3 3 3 103 ];
[3] Bond3 = [ 15 15 15 15 115 ];
[4] Bond4 = [ 15 15 15 15 15 15 15 15 15 115 ];
[5] pt1 = [1:5]';
[6] pt2 = [1:10]';
[7] P1 = 0; P2 = 0; P3 = 0; P4 = 0;
[8] for ( j=1:300 )
[9]     y = j/1000;
[10]     D1 = (1./(1+y).^pt1).*ones(5,1);
[11]     D2 = (1./(1+y).^pt2).*ones(10,1);
[12]     P1(j,1) = Bond1*D1;
[13]     P2(j,1) = Bond2*D2;
[14]     P3(j,1) = Bond3*D1;
[15]     P4(j,1) = Bond4*D2;
[16] end
[17] xx = [0.1:0.1:30]';
[18] figure
[19] plot(xx,P1,'k')
[20] hold on; plot(xx,P2,'k:')
[21] hold on; plot(xx,P3,'k')
[22] hold on; plot(xx,P4,'k:'), xlabel('Rate %'), ylabel('Price')
```

In the script, lines [1]-[7] provide various definitions. First, lines [1]-[4] establish the cashflows of four bonds: Bond1, Bond2, Bond3, and Bond4. Bond1 and 2 have coupons of 3% and Bonds 3 and 4 have coupons of 15%. All four bonds pay annual coupons. Second, Bond1 and 3 are five year bonds while bonds 2 and 4 are ten year bonds. Lines[8]-[16] contain the loop that derives bond prices for varying levels of the yield. The counter-variable j runs from 1 to 300 in steps of 1. When the yield y is calculated in line[9] it is set equal to j divided by 1000. This means that prices are calculated for yields between 0.1% to 30%. Lines[10]-[11] calculate the annual discount factors using the auxiliary variables from lines[5]-[6]. The prices for the four

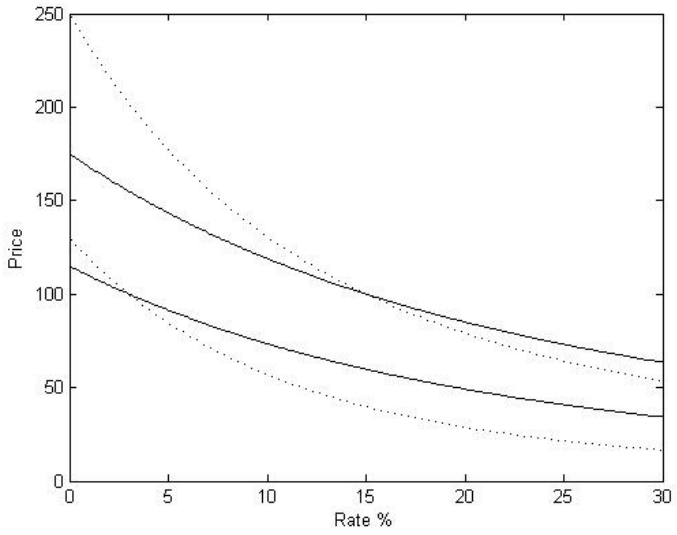


Figure 4.4: Bond price sensitivity to yield levels

different bonds are calculated in lines[12]-[15] following (3.1). The remainder of the script plots the results as shown in Figure 4.4.

Each line in Figure 4.4 shows the bond price as a function of the yield. The dotted lines in the figure correspond to the long bonds (i.e. the ten year bonds) while the full lines correspond to the short bonds i.e. the five year bonds. It is noted that the 3% coupon bonds attain a value of 100 when the yield is 3%, and the 15% coupon attain similarly the value of 100 when the yield is 15%. This is according to theory. It is further noted that the slope of the price-yield relationship is steeper for the longer (i.e. ten year bonds) than it is for the shorter five year bonds; although it can be seen that the difference in slope wears-off as the yield level increases. In addition, it can be seen that the price-yield relationship is inverse so that the price increases when the yield decreases and vise-versa. Hence, the price-yield relationship has a negative slope. Finally it is noted that this relationship is convex and the convexity is higher for the longer bonds than for the shorter bonds.

Based on the inspection of Figure 4.4 as iterated above, it seems reasonable to rely on a linear and a quadratic term to approximate the price change

that follows from a small change in the yield.[6] Such an approximation can be achieved via a Taylor series expansion of order 2 around the bond price P at the interest rate r:

$$\begin{aligned} P(r+\Delta r) &= P(r) + \frac{dP}{dr} * \Delta r + \frac{1}{2}\frac{d^2P}{dr^2} * \Delta r^2 + O \\ \frac{\Delta P}{P} &\approx \frac{1}{P}\frac{dP}{dr} * \Delta r + \frac{1}{2}\frac{1}{P}\frac{d^2P}{dr^2} * \Delta r^2, \end{aligned} \tag{4.9}$$

where the second line follows after subtracting $P(r)$ on both sides of the equation, dividing through by $P(r)$ to attain an expression for relative price changes, and assuming that the remainder O is small. We can now define duration as:

$$D \equiv -\frac{1}{P}\frac{dP}{dr}, \tag{4.10}$$

and convexity as:

$$Conv \equiv \frac{1}{P}\frac{d^2P}{dr^2}. \tag{4.11}$$

On the basis of (4.9) an approximation for the relative bond price change can then be written as:

$$\frac{\Delta P}{P} = -D * \Delta r + \frac{1}{2} * Conv * \Delta r^2.$$

Modified duration is defined as the price sensitivity measure based on the changes in the *yield*, hence deriving dP/dy instead of dP/dr on the basis of (3.5) or (3.6) gives accordingly:[7]

[6] Strictly speaking, the causality is reversed here. Previously in the text it was mentioned that the yield can be calculated only when the bond price is known. What we do now is to approximate price changes that follow from yield changes. This is legitimate if we treat the yield curve as a general market measure and then derive prices for bonds other than those that were used to derive the yield curve. In practise, we will often reason on the basis of yields and use yield curve movements to find the bond prices that interest us.

[7] Differentiate expression (3.5) with respect to y gives (4.12), and differentiating (3.6) with respect to y gives (4.13).

$$MD = \frac{1}{P} * \frac{1}{1+y} * \left[\sum_{j=1}^{n} j * \frac{C}{(1+y)^j} + n * \frac{100}{(1+y)^n} \right] \quad (4.12)$$

$$= \frac{1}{P} * \left\{ \frac{C}{y^2} * \left[1 - \frac{1}{(1+y)^n} \right] - \left[100 - \frac{C}{y} \right] * \frac{n}{(1+y)^{n+1}} \right\} \quad (4.13)$$

where the first expression follows from (3.5) and the second from (3.6). An approximation formula to the modified duration of a bond can be derived from (4.13) when assuming that the bond under investigation is a par bond[8], i.e. assuming that $C = 100 * y$ and that $P = 100$.[9]

$$MD \approx \frac{1}{100} \left\{ \frac{100 * y}{y^2} * \left[1 - \frac{1}{(1+y)^n} \right] - \left[100 - \frac{100 * y}{y} \right] * \frac{n}{(1+y)^{n+1}} \right\}$$

$$= \frac{1}{y} \left[1 - \frac{1}{(1+y)^n} \right]. \quad (4.14)$$

Similarly to the above derivation of the modified duration for a bond by the first derivative of the pricing expression with respect to the yield, the expression for (modified) convexity ($MConv$) can be found by differentiating the expression for dD/dy with respect to y:

[8] Note that $\frac{1}{y}\left[1 - \frac{1}{(1+y)^n}\right] = \frac{1}{y}\left[1 - (1+y)^{-n}\right] = \frac{1/(1+y)}{y/(1+y)}\left[1 - (1+y)^{-n}\right]$
$= \frac{1-(1+y)^{-n}}{(1+y-1)/(1+y)}(1+y)^{-1} = \frac{1-(1+y)^{-n}}{\frac{1+y}{1+y} - (1+y)^{-1}}(1+y)^{-1}$
$= \frac{1-(1+y)^{-n}}{1-(1+y)^{-1}}(1+y)^{-1}$
This expression has the form sometimes encountered in text books. For example, Campbell, Lo, and MacKinlay (1997, p408) presents an approximation to the bond duration, where duration is expressed as the Macaulay's duration. Remembering that $D_{Macaulay} = (1+y) * MD$, makes it clear that dividing the expression derived by Campbell, Lo and MacKinley with $(1+y)$, constitutes a transformation from Macaulay duration to modified duration and is thus identical to (4.14).

[9] Note that (4.14) does not constitute an approximation for par bonds. In the case of par bonds (4.14) holds exactly.

$$
\begin{aligned}
MConv &= \frac{1}{P} * \frac{1}{(1+y)^2} \left[\sum_{j=1}^{n} j*(j+1)*\frac{C}{(1+y)^j} \right. \qquad (4.15)\\
&\quad \left. +n*(n+1)*\frac{100}{(1+y)^n} \right] \\
&= \frac{1}{P} * \left\{ \frac{1}{(1+y)^2} \left[\frac{(n+1)*n}{(1+y)^n} \left(100 - \frac{C}{y} \right) \right. \right. \\
&\quad \left. \left. -\frac{2*C}{(1+y)^{n-2}} \left(\frac{1}{y^3} + \frac{n}{(1+y)*y^2} \right) \right] + \frac{2*C}{y^3} \right\} \qquad (4.16)
\end{aligned}
$$

A par-bond approximation to the convexity expression is obtained by assuming that $C = 100 * y$ and that $P = 100$, similar to the derivation of (4.14) above. When inserted in (4.16) these assumptions give the following approximative formula:

$$MConv \approx \frac{2}{y^2} * \left[1 - \frac{1}{(1+y)^n} - \frac{n*y}{(1+y)^{n+1}} \right]. \qquad (4.17)$$

The above derived expressions for the modified duration and convexity of a bond leads to the following approximation for bond price changes on the basis of changes in the *yield*:

$$\frac{\Delta P}{P} = -MD * \Delta y + \frac{1}{2} * Conv * \Delta y^2. \qquad (4.18)$$

The formulas presented above for calculating the bond price (3.6), the modified duration (4.13) and the convexity (4.16) are implemented in the Matlab(R) function shown below:

```
[1] function [out] = bond(in)
[2]%
[3]% Chapter:  Risk and Return
[4]%
[5]% This function calculates the price, modified duration
[6]% and convexity of a coupon bond having annual payments
[7]%
[8]% Usage:
```

```
[9]% [out] = bond(in)
[10]%
[11]% in (structure):
[12]% .Y - yield of the bond 1-by-1 e.g.  0.053 = 5.3%
[13]% .C - coupon of the bond 1-by-1 e.g.  5 (=5% bond)
[14]% .N - years to maturity 1-by-1 e.g.  2.5 (=2 years and 6 months)
[15]% out (structure):
[16]% .P - Price of the bond
[17]% .MD - Modified duration of the bond
[18]% .Conv - Convexity of the bond
[19]%
[20]% date:  November 2006
[21]% report bugs to:  email@kennyholm.com
[22]%
[23] y    = in.Y; c = in.C; n = in.N;
[24] p    = c/y*( 1-1/(1+y)^n ) + 100/(1+y)^n;
[25] md  = 1/p*( c/y^2*(1-1/(1+y)^n) - n/(1+y)^(n+1)*(100-c/y) );
[26] conv = (1/p)*(1/(1+y)^2*( (n+1)*n/(1+y)^n*(100-c/y) ...
[27]           - 2*c/(1+y)^(n-2)*(1/y^3 + n/((1+y)*y^2)) ) + 2*c/y^3);
[28] out.P=p; out.MD=md; out.Conv=conv
```

The help text in the preamble of the function (lines[2]-[22]) outlines the input data required to run the function and specifies the outputs that the function produces. Line [1] contains the function name "bond" together with the input structure called *in* and the output structure called *out*. In line [23] the input values are redirected to local variables. This is done for convenience only. Line [24] implements equation (3.6), line [25] implements equation (4.13), and finally, lines [26]-[27] implement equation (4.16). Line [28] concludes the function by assigning the results to the output structure.

Using the above *bond* function it is possible to analyse how well the performance is of the approximative formulas for the modified duration and convexity (4.14) and (4.17) respectively. Such an analysis is generated in the below Matlab(R) script file.

```
[1] max_ = 100;    % the maximum number of time periods to investigate
[2]% ...  assigning the input parameters to IN structure
[3] X.Y = 0.06;    % yield
[4] X.C = 5.5;     % coupon
```

```
[5] X.N = 0;       % initial number of years
[6]% ...  generating containers for the results
[7] md         = zeros(max_,1);
[8] md_approx  = zeros(max_,1);
[9] conv       = zeros(max_,1);
[10] conv_approx   = zeros(max_,1);
[11] conv_approx_1 = zeros(max_,1);
[12] p = zeros(max_,1);
[13] y = X.Y;
[14] c = X.C;
[15] for ( j=1:max_ )
[16]    X.N=j;
[17]    [out] = bond(X);     % calling the bond function
[18]    p(j,1) = out.P;       % assign outputs
[19]    md(j,1) = out.MD;     % assign outputs
[20]    conv(j,1) = out.Conv; % assign outputs
[21]    md_approx(j,1) = 1/y*(1-1/(1+y)^j);
[22]    conv_approx(j,1) = 2*y/y^3*(1-1/(1+y)^j-y*j/(1+y)^(j+1));
[23] end
[24]% ...  plotting results
[25] figure
[26] subplot(2,1,1), plot(md,'k-'), xlabel('Bond maturity'), ylabel('Modified duration')
[27] hold on
[28] plot(md_approx,'k--')
[29] subplot(2,1,2), plot(conv,'k-'), xlabel('Bond maturity'), ylabel('Convexity')
[30] hold on
[31] plot(conv_approx,'k--')
```

The code above is well annotated so there is no need to go through it here in detail. The results generated by the script are shown in Figure 4.5. The solid lines in the figure show the exact formulas and the dotted lines the approximative formulas. It can be seen that he approximative formulas perform relatively well for the chosen parameters.

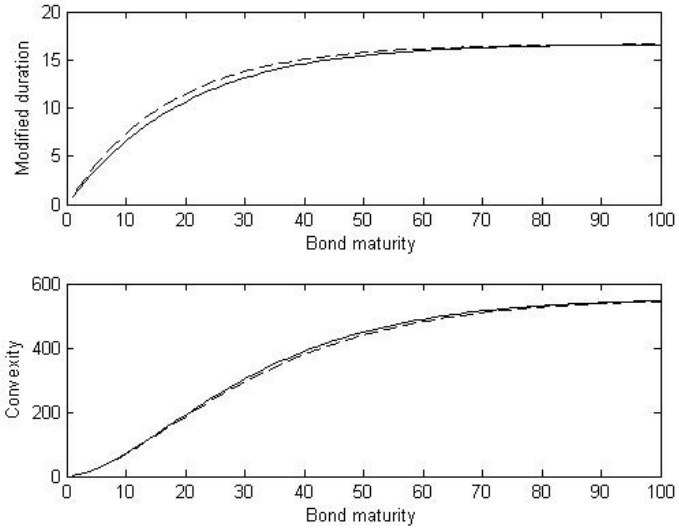


Figure 4.5: Approximations to the modified duration and convexity

4.3 Fixed Income Returns

Financial asset prices fluctuate over time, however, on average they tend to increase in nominal as well as real terms. There are two main reasons for this. Firstly, most humans prefer to consume today rather than tomorrow, or at any other future point in time. Hence, a compensation must be obtained by investors for postponing their consumption to a future date.[10] Secondly, for almost all assets the future pay-off is unknown; this is for example the case for equities and for bonds sold before maturity. Consequently, as the prices of these assets fluctuate as times goes by, as an unknown function of underlying market factors, the future pay-off to investors is uncertain / risky. Most investors (and humans) have aversion against uncertainty and thus require a positive compensation for assuming such uncertainty / risk. For these reasons, in financial theory and investment management, the return and risk on an asset is synonymous with the *expected* return on the asset.

[10]This compensation is paid by the counterparty in the financial market. If for example a firm needs capital to fund its investments in machinery, it can raise money by issueing bonds in the capital markets. These bonds offer an interest rate which is paid by the firm to make it worthwhile for investors to postpone their comsumption.

Likewise, since the risk of an asset is unobservable and we need to estimate it, we also talk about the *expected* risk.

Let t denote time and E the expectation operator, then the total or gross return on an asset can be defined by:

$$E\left(r_t^{tot}\right) = \frac{\text{future amount received}}{\text{amount paid}} = \frac{E\left(P_t\right)}{P_{t-1}} \tag{4.19}$$

and the relative return can be defined as:

$$\begin{aligned} E\left(r_t\right) &= \frac{\text{future amount received - amount paid}}{\text{amount paid}} \\ &= \frac{E\left(P_t\right) - P_{t-1}}{P_{t-1}} = \frac{E\left(\Delta P_t\right)}{P_{t-1}}. \end{aligned} \tag{4.20}$$

It is clear that:

$$E\left(r_t\right) = \frac{E\left(P_t\right)}{P_t} - \frac{P_{t-1}}{P_{t-1}} = E\left(r_t^{tot}\right) - 1$$

and that:

$$E\left(P_t\right) = \left[1 + E\left(r_t\right)\right] * P_{t-1.}$$

A continuos compounding version of the above return calculation also exists. According to this the return is calculated by:

$$E\left(r_t\right) = \log\left(\frac{E\left[P_t\right]}{P_{t-1}}\right).$$

To ease notation, in the following the expectation operator E is suppressed. It is thus assumed, unless otherwise stated, that the future pay-off, risk and return are unknown and only expectations to these entities can be formed. Actually, much of the "art" of investment management originates from devising models and techniques to efficiently and as precisely as possible estimate the future unknown returns for the eligible investment universe that one deals within.

The return from holding an equity can be broken down into the following components:

$$r^{equity} = \text{change in price} + \text{dividends},$$

where both parts are difficult to estimate. The first term typically depends on the general evolution of the equities markets, the state of the economy, the state of the industry sector to which the firm belongs, as well as market specific factors prevailing at the time of the sale. The second term is also difficult to proxy: even though companies in general publicise their intended dividend policy, it is far from certain that they also stick to it. Hence, both terms in the equity return calculation are stochastic and fairly difficult to approximate.

The holding period return of a bond comprises the following elements:

$$r^{bond} = \text{change in price} + \text{coupons} + \text{passage of time.} \tag{4.21}$$

The first element of the bond return in (4.21) can be proxied by (4.18), hence:

$$change\ in\ price = -MD * \Delta y + \frac{1}{2} * Conv * \Delta y^2. \tag{4.22}$$

Coupons, relative to the price, received over a given interval of time is defined by:

$$coupons = \frac{C}{P} * \Delta t, \tag{4.23}$$

where the time interval Δt matches the time interval over which the yield change is calculated in (4.22) and is given as a fraction of the coupon period.[11] The last part of (4.21) concerns the passage of time. To gauge this components consider a bond at time t having maturity n. At time $t+1$ this bond will have a maturity equal to $n-1$. Hence, all else equal, the approximation to the passage of time can be done by calculating the price difference between the same bond at time $t+1$ and t using e.g. (3.6). It is the intention of this calculation to derive an expression akin to $\frac{\partial P}{\partial t}$:

[11] Calculation of bond prices that ignore this accrued coupon payment are referred to as "clean prices", whereas when the accrued coupon part is included in the prices, they are referred to as "dirty prices".

$$\begin{aligned}
\frac{P_{t+1}-P_t}{P_t} &= \frac{1}{P_t} * \left\langle \frac{C}{y} * \left[1 - \frac{1}{(1+y)^{n-1}}\right] + \frac{100}{(1+y)^{n-1}} \right. \\
&\qquad \left. - \left\{ \frac{C}{y} * \left[1 - \frac{1}{(1+y)^{n}}\right] + \frac{100}{(1+y)^{n}} \right\} \right\rangle \\
&= \frac{1}{P_t} * \left\langle \frac{C}{y*(1+y)^{n}} - \frac{C}{y*(1+y)^{n-1}} + \frac{100}{(1+y)^{n-1}} - \frac{100}{(1+y)^{n}} \right\rangle \\
&= \frac{1}{P_t} * \left\langle \frac{C}{y\,(1+y)^{n}} - \frac{C*(1+y)}{y\,(1+y)^{n}} + \frac{100*(1+y)}{(1+y)^{n}} - \frac{100}{(1+y)^{n}} \right\rangle \\
&= \frac{100*y-C}{P_t*(1+y)^{n}}. \qquad (4.24)
\end{aligned}$$

Combining (4.22), (4.23) and (4.24) gives an approximation to the bond return[12]:

$$r^{bond} = \underbrace{-MD*\Delta y + \frac{1}{2}*Conv*\Delta y^2}_{change\ in\ price} + \underbrace{\frac{C}{P}*\Delta t}_{coupon} + \underbrace{\frac{100*y-C}{P*(1+y)^{n}}}_{passage\ of\ time} * \Delta t \qquad (4.25)$$

As seen above, the underlying factor that determines bond prices, and hence bond returns, is the yield curve's shape and location, see e.g. equation (3.1) or (3.6). For bonds where the issuer is assumed to be default free, i.e. always capable of paying coupons, the coupon part above is known with certainty. It is often assumed that government bonds are default free. For corporate bonds, i.e. bonds issued by firms rather than governments, it can be reasonable to assume that the coupon and terminal payment are uncertain. However, typically these payments will be significantly more certain than the dividend payments of the equity return. Firstly, the coupon payments are promised payments, unlike dividends. Dividends are paid if there are funds available; coupons have to be paid in the same way as interest payments on a loan from the bank; if bond holders do not get their coupons on time they can declare the firm in bankruptcy. Secondly, in the event of a bankruptcy, bond

[12]The yield is denomibated as a fraction, e.g. $y = 0.05$ if the yield is 5%, whereas the coupon is denomined in a monetary unit, e.g. $C = 5$, if the coupon rate is 5%. In effect, the 100 in the numerator of the expression for the passage of time (4.24) serves to make the yield and the coupon comparable in sizes.

holders will typically receive part of their money back: the sale of the firm's assets will be used to pay a fraction of the bond holders claims, while equity holders get nothing (the bankruptcy would be unjustified if equity holders were to get anything). The third term in the bond return decomposition is *passage of time*. This term refers to the property that the bond price converges to its terminal value (pull-to-par effect) when the time to maturity shortens. This term is non-stochastic.

Equation (4.25) constitutes an approximation to the return on a default free coupon paying bond. The exact calculation expression taking into account the factors mentioned by (4.21) can be written as:

$$r_{t,t+j}^{bond} = \frac{P_{t+j}^{(n-j)} - P_t^n + C * j}{P_t^n}. \tag{4.26}$$

Here n refers to the number of remaining periods until the bond matures. The non-stochastic pull-to-par effect is accounted for by the maturity shortening of the bond at time $t+1$: the bond is acquired at time t when it has maturity n, as signified by P_t^n. After j periods the holding period return is calculated from t to $t+j$, the bond has come closer to maturing due to the passage of j periods, hence, the price at $t+j$ refers to a bond that has a remaining maturity of $n-j$ periods, as signified by P_{t+j}^{n-j}.

Chapter 5

Term Structure Models

5.1 Introduction

This chapter presents different ways to model yield curves in the maturity as well as in the time-series dimensions. It presents factor models and distinguishes between arbitrage free and not-necessarily arbitrage free models. We will see how the fundamental partial differential equation (PDE) is going to help us in constructing arbirage free yield curve models and how multivariate time-series models help to construct dynamic versions of the not-necessarily arbitrage-free parametric yield curve models.

Learning objectives

- Understand the difference between arbitrage free models and other types of yield curve models
- Know the difference between affine and quadratic models
- Be able to use yield curve models in practise
- Learn how to interpret yield curve factors

5.2 Not-Necessarily Arbitrage Free Models

This section presents a yield curve modelling strategy that is based on econometrics and empirical observations. It basically takes yield curve observations and/or bondprices as given and sets out to model these quantities as well as possible, without forming an opinion on whether observations result from an equilibrium economic model or what the underlying fundamental mechanism are that generate yields and prices. Such a modelling approach is in contrast to the so-called "no-arbitrage models", that in general assume a dynamic evolution for yield curve factors under a risk-neutral measure and explicitly writes down the functional form for the market price of risk, and as such imposes a certain structural form on the economic agents that trade and thus generate price and yield observations. The no-arbitrage modelling framework is presented in Section 5.3. In the sections below we present two models that are not necessarily arbitrage free. These are the Nelson-Siegel and Svenson-Soderlind models. A dynamic represenation of the Nelson-Siegel model is presented by Diebold and Li (2006). It is this dynamic framework that we use below, while we also explain more "traditional" estimation procedures.

5.2.1 Nelson and Siegel

A parametric three-factor specification for the yield curve at a given point in time is suggested by Nelson and Siegel (1987). The three factors they put forth represent the yield level at infinite maturity, the slope (the difference between the yield observed at the long and the short end of the yield curve; and curvature of yield curves, i.e. how much the curve "bends" in the medium maturity spectrum. By using three factors and one time-decay parameter the model proves to explain well the yield curve location and shape as a function of maturity. Another way to put this is to say that the model can capture the major part of the variability of yield changes over time, and is as such akin to a regular factor decomposition of the yield curve (see, e.g. Litterman and Scheinkman (1991))[1]. The difference to a factor analysis is that Nelson and Siegel (1987) prespecifies the factors and a priori assigns a given interpretation to each of the three factors (these being: level,

[1] A factor model (which is similar to a principle component analysis) finds underlying factors in a data set (covariance matrix) in such a way that the factors are uncorrelated and maximise the explanatory power of the extracted factors. For information see, Johnson and Wichern (1992)[ch. 8 and 9].

slope and curvature)[2]. Figure 5.1 offers a visual interpretation of these yield curve factors and Figure 5.2 gives the interpretation of the yield curve factor sensitivities: these two elements correspond to the β_t and H, respectively, in equation (5.1). The sensitivity matrix is to a large extent defining the Nelson-Siegel model, and this is further elaborated in equation (5.2)

Following the intuitive parametrisation suggested by Diebold and Li (2006) the functional form for the Nelson-Siegel model is:

$$Y_t(\tau) = H \cdot \beta_t + e_t, \tag{5.1}$$

where $Y_t(\tau)$ is the vector of yields observed at time t for the T maturities τ, and $e \sim N(0, R)$. The matrix of factor sensitivities is explicitly given by:

$$H = \begin{bmatrix} 1 & \frac{1-\exp(-\lambda\tau_1)}{\lambda\tau_1} & \frac{1-\exp(-\lambda\tau_1)}{\lambda\tau_1} - \exp(-\lambda\tau_1) \\ 1 & \frac{1-\exp(-\lambda\tau_2)}{\lambda\tau_2} & \frac{1-\exp(-\lambda\tau_2)}{\lambda\tau_2} - \exp(-\lambda\tau_2) \\ \vdots & \vdots & \vdots \\ 1 & \frac{1-\exp(-\lambda\tau_T)}{\lambda\tau_T} & \frac{1-\exp(-\lambda\tau_T)}{\lambda\tau_T} - \exp(-\lambda\tau_T) \end{bmatrix}. \tag{5.2}$$

The yield curve factors are collected in $\beta_t = \{\text{level}_t, \text{slope}_t, \text{curvature}_t\}^T$. The variable, λ, determines the speed of time-decay for the slope and curvature sensitivities, i.e. the particular patterns followed by the sensitivities as depicted in Figure 5.2. In theory, there should be a time subscript on λ so that it would become λ_t and thus be allowed to vary over time; however, in practise it only marginally improves the fit of the Nelson-Siegel curve to include such time-dependancy on the account of λ when fitting the Nelson-Siegel curve to yield curves observed at different time-points. The marginally improved fit weighted against the significant increase in the number of estimated parameters it would entail (the number of estimated parameters would increase by $n-1$, where n is the number dates in the sample) makes it in general not worthwhile to add a time-subscript to λ.

As an initial step to get familiar with the Nelson-Siegel model we can consider λ to be a constant, rather than a parameter to estimate. Then (5.1) reduces to a linear regression that can then be solved for different values of λ on the basis of which the optimal λ can be chosen as the one that minimises

[2]Note that Nelson and Siegel (1987) originally formulated the model in terms of forard rates. By integrating the forward rate expressions the yield form of the Nelson-Siegel model appears, as presented in this chapter. The reader is invited to perform the relevant calculations in **??** exercise at the end of this chapter.

the squared errors. The following Matlab code implements such a solution strategy.

```
[1] Y = [2.17 2.47 2.85 3.39 4.01 4.31 ...
[2]   4.48 4.58 4.65 4.70 4.74 4.77 4.79]';
[3] tau  = [1 3 6 12 24 36 48 60 72 84 96 108 120]';
[4] nTau = length(tau);
[5] for ( k=1:50 )
[6]    L = k/200+0.01;
[7]    H = [ones(nTau,1) ...
[8]    (1-exp(-L.*tau))./(L.*tau) ...
[9]    (1-exp(-L.*tau))./(L.*tau)-exp(-L.*tau)];
[10]    Beta = H\Y;
[11]    u = Y-H*Beta;
[12]    stderr = sqrt( diag ((u'*u)/(length(Y)-4)*pinv(H'*H) ));
[13]    Sum_u2 = sum(u.^2);
[14]    Res(k,:)  = [ Sum_u2 Beta' L ];
[15] end
[16] optim = find(Res(:,1)==min(Res(:,1)));
[17] disp('The optimal Nelson-Siegel parameters')
[18] disp(' Level Slope Curvature Lambda')
[19] disp(Res(optim,2:end))
[20] disp(stderr')
```

The following results are produced when the above Matlab(R) code is executed:

```
Optimal Nelson-Siegel parameters & standard errors
 ----------------------------------------
 Level    Slope   Curvature   Lambda
 4.9988  -2.9969   0.49941     0.1
 0.023184 0.065405 0.20141
```

A short description of the code is warranted. Lines [1]-[3] define the yield curve under investigation and at which maturities the yields are observed. Line [4] calculates the number of yield observations in Y: this is needed to construct the H matrix i.e. to get its row-dimension right. The main calculations are performed in the loop covered by lines [5]-[15]. k represents

the counter variables that in line [6] is translated into the value for λ so that $\lambda \in \{0.015, 0.02, \ldots, 0.26\}$. Naturally some form of prior knowledge is necessary in order to determine the appropriate range for λ or the range can be determined by trial-and-error. In lines [7]-[9] the H matrix is defined. In terms of a linear regression H represents the "x variables", i.e. the variables on the right hand side of the regression equation. The LHS of the regression equation is the vector of yields Y and the parameters to be estimated are collected in β. Line [10] conducts the regression, line [11] calculates the residuals, and line [12] calculates the standard errors of β.

As hinted at above, the parameter λ is estimated in a two step procedure, where the first step calculates the regression equation for a series of potential values of λ, then in a second step the λ that provides the best fit is chosen. As a proxy to determine the best fit this example uses the sum of squared residuals (similar to the OLS criterion): line [13] calculates this sum of squared residuals. Line [14] stores the results generated by the loop and line [15] ends the loop. Line [16] finds the row number of the result matrix in which the selection criterion is minimised. Lines [17]-[20] display the results.

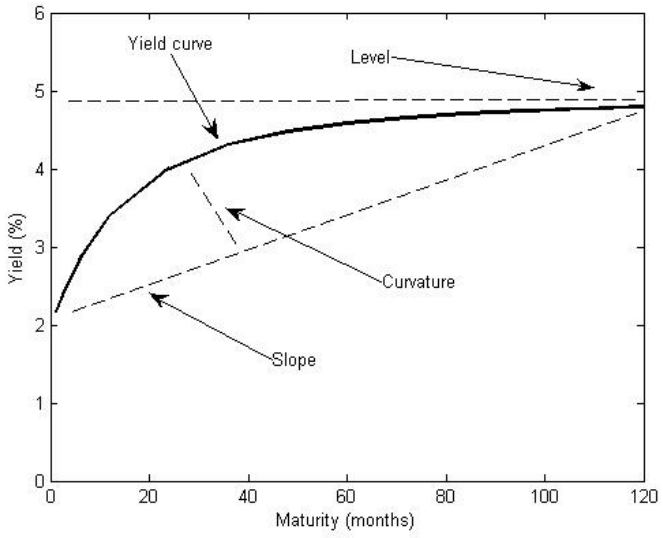


Figure 5.1: Nelson-Siegel yield curve factors

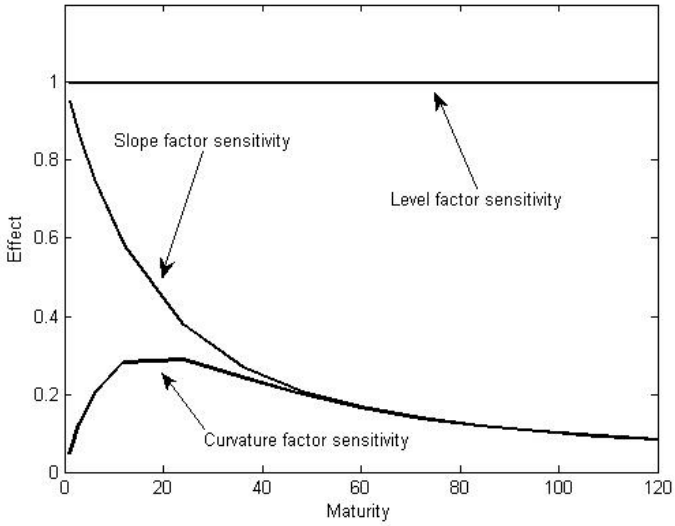


Figure 5.2: Nelson-Siegel factor sensitivities ($\lambda = 0.08$)

5.2.2 Svensson and Soderlind

A four factor extension of the Nelson-Siegel factor loading matrix is suggested by Söderlind and Svensson (1997). In practice, this extension comprises a second curvature factor loading, giving the model additional degrees of freedom to better fit observed yields. A formulation of the Svensson-Soderlind model is given in (5.3).

$$Y_t(\tau) = G \cdot \alpha_t + v_t, \tag{5.3}$$

where

$$G = \begin{bmatrix} 1 & \frac{1-\exp(-\lambda_1\tau_1)}{\lambda_1\tau_1} & \frac{1-\exp(-\lambda_1\tau_1)}{\lambda_1\tau_1} - \exp(-\lambda_1\tau_1) & \frac{1-\exp(-\lambda_2\tau_1)}{\lambda_2\tau_1} - \exp(-\lambda_2\tau_1) \\ 1 & \frac{1-\exp(-\lambda_1\tau_2)}{\lambda_1\tau_2} & \frac{1-\exp(-\lambda_1\tau_2)}{\lambda_1\tau_2} - \exp(-\lambda_1\tau_2) & \frac{1-\exp(-\lambda_2\tau_2)}{\lambda_2\tau_2} - \exp(-\lambda_2\tau_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \frac{1-\exp(-\lambda_1\tau_T)}{\lambda_1\tau_T} & \frac{1-\exp(-\lambda_1\tau_T)}{\lambda_1\tau_T} - \exp(-\lambda_1\tau_T) & \frac{1-\exp(-\lambda_2\tau_T)}{\lambda_2\tau_T} - \exp(-\lambda_2\tau_T) \end{bmatrix}. \tag{5.4}$$

Here, α collects the yield curve factors, {level, slope, curvature1, curvature2} and G comprises the assumed factor loading matrices. It is straight forward

to adapt the Matlab(R) code presented in Section 5.2.1 to fit the Svensson-Soderlind model.

```
[1] Y = [2.17 2.47 2.85 3.39 4.01 4.31 ...
[2]  4.48 4.58 4.65 4.70 4.74 4.77 4.79]';
[3] tau = [1 3 6 12 24 36 48 60 72 84 96 108 120]';
[4] nObs = length(Y);
[5] z = 1;
[6] for ( j=1:50)
[7]    L2 = j/200+0.01;
[8]    for ( k=1:50 )
[9]       L1 = k/200+0.01;
[10]       G = [ones(nObs,1) ...
[11]               (1-exp(-L1.*tau))./(L1.*tau) ...
[12]               (1-exp(-L1.*tau))./(L1.*tau)-exp(-L1.*tau) ...
[13]               (1-exp(-L2.*tau))./(L2.*tau)-exp(-L2.*tau)];
[14]       alpha = G\Y;
[15]       u = Y-G*alpha;
[16]       stderr = sqrt( diag ((u'*u)/(length(Y)-4)*pinv(G'*G) ));
[17]       Sum_u2 = sum(u.^2);
[18]       Res(z,:)  = [ Sum_u2 alpha' L1 L2 ];
[19]       z=z+1;
[20]    end
[21] end
```

Compared to the Nelson-Siegel code, an additional loop is added to cater for the additional curvature time-decay parameter to be estimated, see line [6]. Naturally, also the factor sensitivity matrix, G, is adapted in line [13] according to equation (5.4).

5.3 Arbitrage Free Models

Lets start with a disclaimer! It is well beyond the scope of the present book to give an in-depth treatment of arbitrage-free pricing and corresponding yield curve model construction. There are many excellent books that already cover this area such as, among others, Rebonato (1998), Brigo and Mercurio (2001), Bjork (2004), Cairns (2004) and James and Webber (2000). The approach

taken in the current exposition is instead very applied indeed. We start by assuming the existence of two general "axioms" that help us construct arbitrage-free interest rate modelling: we have full faith in these axioms and do not question them, even if we do not fully understand where they come from and their deeper meaning. Based on this foundation, we can then procede and derive yield curve modelling expressions for some of the classical no-arbitrage yield curve models and implement them in Matlab.

Axiom 1: The price P of a bond at time t with maturity τ can be written as:

$$P(\tau)_t = \exp\left[A\left(\tau\right) + B\left(\tau\right) * X_t\right], \tag{5.5}$$

where A and B are functions of the maturity τ, and X_t collects the factors that explain bond prices / yields. A and B can be In this section we will assume that there is only one yield curve factors X and that it is equal to the short rate, i.e. the interest rate for $\tau = 0$. Hence, we set $r_t = X_t$. An implication of Axiom 1 is that the yield-to-maturity can be written as:

$$y\left(\tau\right)_t = -\frac{1}{\tau} * \left[A\left(\tau\right) + B\left(\tau\right) * X_t\right], \tag{5.6}$$

if yields are defined as:

$$y\left(\tau\right) = \frac{-\log\left[P\left(\tau\right)\right]}{\tau}.$$

Axiom 2: It is assumed that the process followed by the short rate can be written as

$$dr = m\left(r\right) * dt + s\left(r\right) * dz,$$

where d represents the first difference operator in continuous time, in discrete time this would equal Δ. So, we are expressing how the changes in r evolve over a short time inteval, dt. The variable $m\left(r\right)$ is the mean of the process, which is allowed to depend on the level of the short rate at time t, and $s\left(r_t\right)$ is the volatility of the process, which like the mean, can depend on the short rate level. And, Z is a Brownian motion, i.e. the generator of random noise around the mean path followed by dr, where $\Delta z = e * \sqrt{\Delta t}$, and $e \sim N\left(0, 1\right)$.

In the absence of arbitrage it can be shown that the fundamental partial differential equation (PDE), which the bond price must obey, can be written

as:[3]

$$\frac{1}{2}\frac{\partial^2 P}{\partial r^2}s^2 + \frac{\partial P}{\partial r}\{m - \pi * s\} + \frac{\partial P}{\partial t} - rP = 0, \tag{5.7}$$

where the price of the bond at maturity is assumed to pay the principal, hence $P(T) = 1$, and π denotes the market price of risk. In order to use (5.7) we need to be able to calculate the first and second partial derivatives of P with respect to the short rate r, and the the first partial derivative of P with respect to time, t. Matlab(R) can be used to validate the derivatives we find when doing the calculations by hand. To this end the symbolic calculation feature can be used.

```
[1] syms r P A B t;
[2] P = exp(A+B*r);
[3] disp('first derivative of P with respect to r')
[4] diff(P,r)
[5] disp('second derivative of P with respect to r')
[6] diff(diff(P,r))
```

Where line [1] defines the relevent variables to be used in the symbolic calculations. Line [2] defines the pricing function and the remaining part of the displayed code calculated first and second derivatives of the pricing equation with respect to the short rate. The calculation for the first derivative of the price with respect to time is not shown in the above matlab code. Recognising that t and τ are inversely related i.e. as time progresses the maturity of the instrument is reduced and applying the chain rule for differentiation (see annex) will allow us to calculate the time-differential of P. The calculated derivatives are shown below:

$$\begin{aligned}
\frac{\partial P}{\partial r} &= B(\tau) * P \\
\frac{\partial^2 P}{\partial r^2} &= B(\tau)^2 * P \\
\frac{\partial P}{\partial t} &= \left[\frac{dA(\tau)}{d\tau} + \frac{dB(\tau)}{d\tau} * r\right] * -P.
\end{aligned}$$

[3]The derivation of the fundamental pricing PDE can be done using Ito's lemma and a hedge arguments between two bonds with different maturities, see Bjork (2004, Section 21.1)

These derivatives are substituted into (5.7) to give:

$$\frac{1}{2}*B(\tau)^2*P*s^2+B(\tau)*P*\{m-\pi*s\}-\left[\frac{dA(\tau)}{d\tau}+\frac{dB(\tau)}{d\tau}*r\right]*P-rP=0$$

where it is seen that P enters in all the terms. Dividing by P and re-organising gives the following equation:

$$\frac{1}{2}*B(\tau)^2*s^2+B(\tau)*\{m-\pi*s\}-\frac{dA(\tau)}{d\tau}-\frac{dB(\tau)}{d\tau}*r-r=0, \quad (5.8)$$

which together with the boundary condition mentioned above, i.e. $P(T)=1$, form the basis for our derivation of a closed form solutions to a classical short rate model as presented below. To this end we use Matlab(R)'s symbolic calculation package, in particluar the function *dsolve*. It is intuitively observed that (5.8) may be split into to ODEs, one containing $\frac{dA(\tau)}{d\tau}$ and another containing $\frac{dB(\tau)}{d\tau}$.[4]

5.3.1 Vasicek

In this section we apply the above illustrated solution technique for the Vasicek short rate model (see, Vasicek (1977) and Rebonato (1998, p.239-242)) with the purpose to derive the corresponding yield curve expression. The following short rate process is assumed:

$$dr=\kappa(\theta-r)\,dt+\sigma dz, \quad (5.9)$$

which is a continous-time version of an autoregressive model with one lag, also called an Orenstein-Uhlenbeck process. This process is mean-reverting around the level θ, and the mean-reversion speed is given by κ. In order to use (5.8) we need to find the appropriate m and s in (5.9). These variables are:

$$\begin{aligned} m &= \kappa(\theta-r), \\ s &= \sigma, \end{aligned}$$

[4]Lund (1998) gives an excellent in-depth presentation of continuous-time yield-curve models and shows in detail how to solve the term-structure equation using the PDE approach. He also presents the solution to some classical models, in particular, Vasicek (1977), Merton (1973) and Cox, Ingersoll, and Ross (1985)

which inserted in (5.8) gives:

$$\frac{1}{2}*B(\tau)^2*\sigma^2+B(\tau)*\{\kappa(\theta-r)-\pi*\sigma\}-\frac{dA(\tau)}{d\tau}-\frac{dB(\tau)}{d\tau}*r-r=0. \quad (5.10)$$

We solve this eqution using Matlab(R)'s symbolic toolbox.

```
[1] syms f B A s k m p r t DA DB
[2] f1 = 0.5*B^2*s^2+B*(k*(m-r)-p*s)-DA-DB*r-r
[3] collect(f1,r)
[4] (-B*k-DB-1)*r+1/2*B^2*s^2+B*(k*m-p*s)-DA
```

Line [1] defines the symbolic variables needed. Note that the variables in (5.10) are translated into Latin counterparts to ease notation and that the dependency of B on τ is suppressed. Line [2] defines expression (5.10) and line [3] isolates the underlying factor, r. We have now the two ordinary differential equations needed and the closed form yield equation for the Vasicek model can be found using *dsolve.*

```
[1] dsolve('-B*k-DB-1=0','B(0)=0')
[2] -1/k+exp(-k*t)/k
[3] B    = -1/k+exp(-k*t)/k
[4] f2   = 1/2*B^2*s^2+B*(k*m-p*s)-DA
[5] dsolve('1/2*(-1/k+exp(-k*t)/k)^2*s^2+(-1/k+exp(-k*t)/k)*(k*m-p*s)-DA=0','A(0)=0')

[6] 1/2/k^2*s^2*t+1/k^3*s^2*exp(-k*t)-1/4/k^3*s^2*exp(-2*k*t)-m*t+...
[7]               1/k*p*s*t-1/k*exp(-k*t)*m+1/k^2*exp(-k*t)*p*s+...
[8]               1/4*(-3*s^2+4*k^2*m-4*k*p*s)/k^3
```

Line [1] defines the ODE for the first part of the problem involving the term $B(\tau)$, and line [2] gives the answer. In line [3] we set the solution equal to B. Line [4] substitutes the solution from the first ODE into the second ODE and line [5] solves it. Lines [6] and [7] contain the somewhat messy solution as found by Matlab(R).[5] After some manipulation of this expression, the yield curve equation for the Vasicek model can therefore be summarised as:

[5] Writing "pretty(dsolve(...))" gives a solution that is a bit easier to read.

$$
\begin{aligned}
y(\tau)_t &= -\frac{1}{\tau} * [A(\tau) + B(\tau) * r_t], \qquad (5.11)\\
B(\tau) &= \frac{e^{-\kappa\tau} - 1}{\kappa},\\
A(\tau) &= B(\tau) * \left[\frac{\pi * \sigma}{\kappa}\right] - \theta * \tau + \frac{\pi * \sigma * \tau}{\kappa} +\\
&\quad \frac{\sigma^2}{4\kappa} * \left[-B^2(\tau) + \frac{2 * e^{-\kappa * \tau} - 2}{\kappa^2} + \frac{2 * \tau}{\kappa}\right]. \qquad (5.12)
\end{aligned}
$$

The following Matlab(R) script implements the above solution.

```
[1] tau  = (1:1:120)';
[2] nTau = length(tau);
[3] p=0.20; s=0.15; k=0.30; m=5; r=4;
[4] B = (exp(-k.*tau)-1)./k;
[5] A = B.*(((p*s)/k)-m) - m.*tau + (p.*s.*tau)./k +(s^2/(4*k)).*( -(B.^2)...
[6]                               + (2*exp(-k.*tau)-2)./(k^2) + (2.*tau)./k );
[7] y = -(1./tau).*(A+B.*r);
```

Line [1] decides the maturity span for the calculated curve. There monthly maturities are chosen having a step length of one month and a maximum maturity of 10 years. Line [3] specifies the input parameters. Lines [4] and [5]-[6] implements the formulas derived above and finally, line [7] calculates the yield curve and stores it in the variable y.

5.3.2 Multi-factor models: an example

A major advantage of the no-arbitrage yield curve models is that they, by construction, ensure consistency between the time-series evolution of the short rate and the shape and location of today's yield curve. Loosely speaking, this consistency embodies the no-arbitrage restrictions. Consistency between short rate dynamics and the shape and location of the yield curves is not necessarily ensured by all yield curve models. In this connection it is instructive to have another look at, e.g. the Nelson-Siegel yield curve model. When re-examining this model, it is clear that it is completely silent on the relation ship between the yield curve shape and the short rate's time-series evolution, or in general, the time-series evolution of the underlying yield curve fators. If

the Nelson-Siegel model is set in a dynamic model context, as e.g. suggested by Diebold and Li (2006), this is even more evident. First we repeat the yield curve equations (5.1) and (5.2) from the above section:

$$Y_t(\tau) = H \cdot \beta_t + e_t, \quad \text{and} \tag{5.13}$$

$$H = \begin{bmatrix} 1 & \frac{1-\exp(-\lambda\tau_1)}{\lambda\tau_1} & \frac{1-\exp(-\lambda\tau_1)}{\lambda\tau_1} - \exp(-\lambda\tau_1) \\ 1 & \frac{1-\exp(-\lambda\tau_2)}{\lambda\tau_2} & \frac{1-\exp(-\lambda\tau_2)}{\lambda\tau_2} - \exp(-\lambda\tau_2) \\ \vdots & \vdots & \vdots \\ 1 & \frac{1-\exp(-\lambda\tau_T)}{\lambda\tau_T} & \frac{1-\exp(-\lambda\tau_T)}{\lambda\tau_T} - \exp(-\lambda\tau_T) \end{bmatrix}, \tag{5.14}$$

and then, in accordance with e.g. Diebold and Li (2006) a certain time-series evolution is hypothesised for the three-yield curve factors collected in β:

$$\beta_t = f\left(\beta_{t-j}, Z_t\right) \approx k + F * \beta_{t-1} + v_t. \tag{5.15}$$

This equation simply states that the time-series evolution of β can be modelled as a function of lagged β's and exogenous factors, for example, as a function of a constant k and β_{t-1}. The actual function form is not important here; what is important is that the parameters in (5.15) do not enter in equations () and (). This illustrates the lack of consistency in the Nelson-Siegel model, and in all other not–necessarily arbitrage-free yield curve models, between the time-series evolution of the underlying yield curve factors and the shape of the yield curve.

Turning to the Vasicek model, we see no similar disconnect. In fact, it is seen that the no-arbitrage consistency is hard-coded into the model, since the parameters that describe the evolution of the short-rate in (5.9), i.e. κ, θ, σ, and the market-price of risk π, all enter the yield curve equation (5.11).

The Vasicek model is an arbitrage-free one-factor model, and the single factor modelled is the short-rate. The Nelson-Siegel model is a three factor model that is not necessarily arbitrage free. Naturally, a three factor model is more flexible than a one-factor model, and can thus better capture relevant features of yield curves than a one factor model can. It is shown by e.g. Duffie and Kan (1996), Dai and Singleton (2000), and a large body of text books of which some are cited above, how one can obtain multi-factor yield-curve models that are arbitrage free. One common feature of this modelling philosophy is that the yield curve factors are treated as being un-observed and

hence needs to be estimated or they are found directly using e.g. the approach of Chen and Scott (1993). Nonwithstanding this, it can be an arduous task to estimate such no-arbitrage multifactor models. A useful alternative is to derive a no-arbitrage version of the Nelson-Siegel class of model as it is done by Christensen, Diebold, and Rudebusch (2007). Another way to incorporate no-arbitrage restrictions on the yield curve and its dynamic evolution over time, is by assuming that the yield curve factors are observable. Then it is possible to apply a two-step estimation procedure as illustrated by Ang, Piazzesi, and Wei (2006). This procedure is applied below using Nelson-Siegel yield curve factors; the exposition draws heavily on Coroneo, Nyholm, and Vidova-Koleva (2007).

At the outset we specify the yield curve equation:

$$Y_t(\tau) = a + b \cdot \beta_t + \epsilon_t, \tag{5.16}$$

where β, as in (5.13) represent the Nelson-Siegel yield curve factors. It is noted, however, that (5.16) contains an additional constant term, a, in contrast to (5.13), or put differently, it is implecitly assumed by the Nelson-Siegel yield curve model that $a = 0$. However, in general it can be observed that a and b in (5.16) play the same role as H in the Nelson-Siegel model. The dynamic evolution of the yield curve factors is assumed to be governed by the following first order autoregression model:

$$\beta_t = \mu + \Phi \cdot \beta_{t-1} + u_t, \qquad u_t \sim N\left(0, \Sigma\Sigma^T\right). \tag{5.17}$$

The parameters in (5.16) are defined in the following way:

$$\begin{aligned} a_\tau &= -\frac{A_\tau}{\tau} \\ b_\tau &= -\frac{B_\tau}{\tau}, \end{aligned} \tag{5.18}$$

where, A and B can be found via recursive formulae[6]:

$$\begin{aligned} A_{\tau+1} &= A_\tau + B_\tau^T \cdot (\mu - \Sigma \cdot \pi_0) + \frac{1}{2} B_\tau^T \Sigma\Sigma^T B_\tau - A_1, \\ B_{\tau+1}^T &= B_\tau^T \cdot (\Phi - \Sigma \cdot \pi_1) - B_1^T, \end{aligned} \tag{5.19}$$

[6]Among others, Ang and Piazzesi (2003, appendix A) and Mönch (2006, appendix A) show how these recursive formula can be derived.

and where A and B are determined from what is called the short-rate equation, because it links the evolution of the rate for a short maturity, r, to the yield curve factors in the following way:

$$r_t = a_1 + b_1 \cdot \beta_t + v_t \tag{5.20}$$

In the arbitrage-free model setup the functional form of the market price of risk is explicitly formulated as a linear function of the yield curve factors:

$$\Lambda_t = \pi_0 + \pi_1 \cdot \beta_t \tag{5.21}$$

It is the recursive structure and the fact that A_1 and B_1 depend on the parameters that govern the dynamic evolution of the yield curve factors, as well as the market price of risk, that ensures internal consistency of the model and hence that it excludes arbitrage, as it was the case for the Vasicek model.

In the below Matlab programs we show how this multi-factor no-arbitrage model can be estimated. It is a quite involved process and we do the implementation in small steps. In summary the game plan is the following:

1) extract Nelson-Siegel yield curve factors and standardise them;

2) estimate μ, Φ, and Σ from (5.17);

3) estimate a_1, and b_1 from (5.20);

4) calculate a_τ and b_τ using (5.19), (5.18) and (5.21);

5) iterate over 2, 3, and 4 to minimise the squared residuals from (5.16);

We already know how to solve step (1) from the previous section 5.2.1 on the Nelson-Siegel model. To facilitate easier convergence in the estimation set we also standardise the observable factors by subtracting their mean and divide by the standard deviation. The estimation involved in step (2) will, strictly speaking, first be presented in section 7.2. However, for now it is enough to know that the involved model can be solved using OLS regression. Hence, step (2) can be completed by applying the techniques learned in section 2.6. Also step (3) can be solved using an OLS regression. In the following we will therefore assume that steps (1), (2) and (3) have been performed and the relevant results have been stored in matrices called: beta=Nelson-Siegel factors, m=μ, F=Φ, S=Σ, a1=a_1, and a2=a_2.

By examining the programs called $NA\_est.m$, NA_a_b, and NA_y_optim, that acompany the book, the reader can see how the described no-arbitrage yield curve model can be estimated. $NA\_est.m$ is the main program, which completes step (1), step (2) and step (3). It also performs step (5) by using the Matlab(R) $fmincon$ function to estimate the parameters of the model. In particular the function NA_y_optim is minimised, as seen by the call to fmincon in

```
.....................................................................
[1]% ;
[2]% Estimates a multi-factor No-Arbitrage yield curve model
[3]%   program NA_est.m
[]...
[64][ out_p, fval, exitfflag ] = fmincon(@NA_y_optim, pstart, [], [], [], [], lb, ub,
[], options_);
.....................................................................
```

The function NA_y_optim contains the calculation of (5.16). In order to do this, it calls the function NA_a_b, which calculates the recursions indicated by (5.19). Finally, NA_y_optim returns the squared residuals between the observed and estimated yields, which is the object that is minimised by $fmincon$. An example of the output generated using the above delinated no-arbitrage model is shown in Figure 5.3. This figure shows observed and estimated yields, and the estimation error.

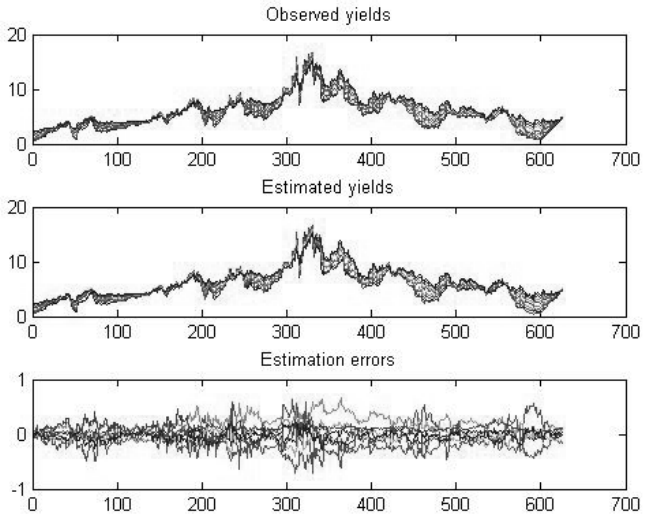


Figure 5.3: Observed and estimated yields from a no-arbitrage model

Chapter 6

Asset Allocation

6.1 Introduction

This chapter presents and applies some of the central ideas of financial theory. The presented tools and techniques aids the process of making informed decisions and to better understand the financial markets. In particular, the chapter shows how to implement the principles of portfolio optimisation in Matlab(R).

Learning objectives

- Understand what diversification is
- Be able to derive efficient frontier portfolios
- Know the basic asset pricing models
- Apply the portfolio principles to fixed income securities

6.2 Efficient portfolios

Mean-variance portfolio theory was developed by Harry Markowitz in the 1950ies.[1] He showed how efficient investment portfolios would form a frontier

[1]The matrial presented in this section draws in particular on Huang and Litzenberger (1988), Luenberger (1998) and **?**.

in the expected return/standard deviation space subject to the following assumptions:

- there are no frictions such as taxes and transaction costs
- investment vehicles are infinitesimally diversible so that fractions of these can be traded
- short-selling is allowed
- investors are concerned only about expected return and risk; the latter being defined as the standard deviation/variance of the expected return
- the returns on the assets in the economy are individually different, and cannot be expressed as linear combinations of other traded assets in the economy; otherwise the covariance matrix of returns would be singular. In other words, the covariance matrix is assumed to be invertible.
- $N \geqq 2$ assets trade in the economy

Let's define r to be the vector of expected returns, C to be the covariance matrix of the asset returns, and w to be the vector of portfolio weights, i.e. how much wealth that is invested in each asset. Much of the work that leads to successful implementation of asset allocation decisions, be it strategic as well as tactical decisions, lies in proper estimation of r and C so that w, the decision variable, can be determined. However, below in the derivation of the mathematics of the efficient frontier r and C are assumed to be known with certainty. The actual estimation of r and C is a separate issue.

Since the investors are concerned only with expected return and risk, the investment portfolios that seem attractive can be represented as a frontier in the expected return - standard deviation space. It is illustrated in the Figure 6.1 that for every level of risk the one portfolio with the highest return is most attractive; similarly, for every level of expected return the portfolio having the lowest standard deviation is most attractive. The efficient frontier that is derived from either of these approaches i.e. either by maximising expected return for all levels of risk, or by minimising risk for all levels of return. Each possible portfolio combination is identified by an "x" in the figure and the efficient frontier by the bold concave line.
The question is now how the portfolio weights w_j can be derived on the basis of r and C; here j counts the observations/portfolios that together

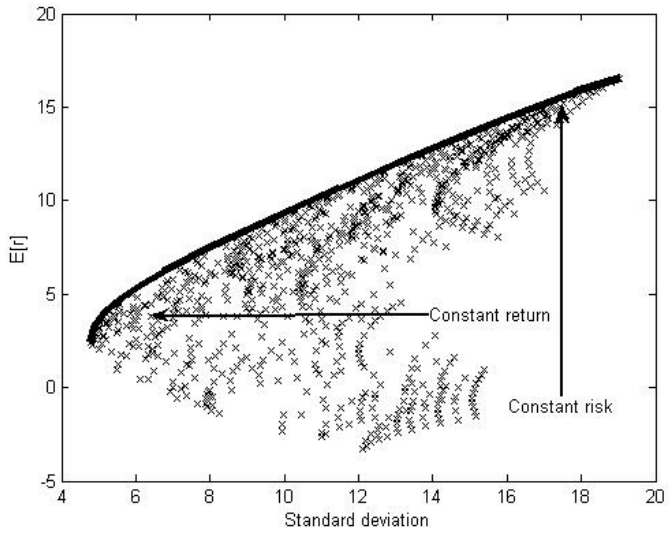


Figure 6.1: Example of efficient and inefficient portfolios

form the efficient frontier. These weights refer to how an investor's capital is distributed between the available investment opportunities in the economy. It is assumed that every investor dedicates a certain portion of her capital to investment purposes; therefore the portfolio weights sum to unity. Or put differently, the portfolio weights are always stated in percentage points or as fractions - they therefore sum to 100%. This idea is also sometimes referred to as the "full investment constraint".

It is a challenge for the investor to choose the portfolio weights wisely. Firstly, this choice depends on the expected returns, standard deviations and covariances that one expects will materialise in the future. Secondly, it depends on the investors risk preference / aversion. In fact, much of financial theory concerns the calculation and choice of these weights, where the choice is assumed to be done in an optimal way. In this connection "optimality" refers to the adherence of the characteristics of the portfolio to the risk-return preferences of the investor.

The two central inputs to the investment process: the expected return of the portfolio and its standard deviation are defined below. The expected return of a portfolio is a weighted average of the instrument returns, where

the weights are defined by the portfolio weights. Hence:

$$r_p = \sum_{n=1}^{N} w_n * r_n = w^T * r. \tag{6.1}$$

The portfolio variance is **not** a weighted average of the asset variances - if it was it would ignore covariance effects i.e. the joint movements of the assets over time. The statistical definition of the variance of two variables is:[2]

$$Var\left(a * X_1 + b * X_2\right) = a^2 * Var\left(X_1\right) + b^2 * Var(X_2) + 2 * a * b * Cov\left(X_1, X_2\right),$$

where the Xs are variables e.g. time-series of returns of two bonds, and a, b are constants. The variance of a portfolio of several variables can the be written as a generalisation of the above:

$$Var\left(r_p\right) = \sigma_p^2 = \sum_{n=1}^{N} \sum_{m=1}^{N} w_n * w_m * \sigma_{n,m} = w^T * C * w \tag{6.2}$$

Expressions (6.1) and (6.2) are stated both with sums and in matrix form. In what follows mainly the matrix form will be used.

As illustrated in Figure 6.1 the optimisation problem faced by investors can be equally well formulated in the two following ways, where the sub-script p stands for "portfolio":

$$\begin{aligned} &\max\ r_p = r^T * w \\ &st. \\ &w^T * C * w = \sigma_p^2 \\ &w^T * \mathbf{1} = 1 \end{aligned}$$

and

$$\begin{aligned} &\min\ \sigma_p^2 = \frac{1}{2} w^T * C * w \\ &st. \\ &w^T * r = r_p \\ &w^T * \mathbf{1} = 1 \end{aligned} \tag{6.3}$$

[2] See e.g.Greene (1993)[p.65].

The first optimisation problem states that the portfolio return should be maximised for a given level of risk, while also fulfilling the full investment constraint i.e. that the sum of the portfolio weights sum to unity; the $\mathbf{1}$ refers to a vector of ones of appropriate dimension. This corresponds to the vertical arrow in Figure 6.1, and it is easy to see that the efficient frontier can be traced out by repeating the optimisation exercise for different levels of portfolio risk σ_p^2. The second optimisation problem corresponds to the horizontal arrow in Figure 6.1 and finds the minimum risk portfolio for a given target level of return while also fulfilling the full investment constraint. The constant in front $\left[\frac{1}{2}\right]$ is added for convenience; when maximising or minimising a function it is naturally the same to optimise $f(x)$ and $k * f(x)$, $k > 0$. It is equally easily seen that a discrete set of points on the efficient frontier can be calculated by varying the target return level. However, Following Markowitz, there exists a more elegant way of characterising the efficient frontier instead of solving the above problems for different target levels of σ_p^2 and r_p. This solution is obtained via the Larange approach in the following way: a) equating the constraints to zero; b) substituting each constraint into the objective function and multiplying each constraint by a positive constant; c) calculating the first derivative of the function with respect to the unknowns and equate the first derivatives to zero; d) then solve the equations for the unknowns. This process is completed below:

$$\min L_{\{w,\lambda,\gamma\}} = \frac{1}{2} w^T * C * w + \lambda \left(r_p - w^T * r\right) + \gamma \left(1 - w^T * \mathbf{1}\right), \qquad (6.4)$$

and the solution is obtained by calculating the first derivative of (6.4) with respect to (w, λ, γ). Since L is composed of three terms, each term can be calculated separately:

$$\begin{aligned}
\frac{\partial L}{\partial w} &= C_{(n,n)} * w_{(n,1)} - \lambda_{(1,1)} * r_{(n,1)} - \gamma_{(1,1)} * \mathbf{1}_{(n,1)} = 0_{(n,1)} && (6.5)\\
\frac{\partial L}{\partial \lambda} &= r_{p_{(1,1)}} - w_{(1,n)}^T * r_{(n,1)} = 0_{(1,1)} && (6.6)\\
\frac{\partial L}{\partial \gamma} &= 1_{(1,1)} - w_{(1,n)}^T * \mathbf{1}_{(n,1)} = 0_{(1,1)}. && (6.7)
\end{aligned}$$

It can be seen via the subscripts showing the dimensions of the included variables that the above list of first derivatives form a $n+2$ system of equations

with $n+2$ unknowns, where n is the number of assets covered by the eligible investment universe. Equation (6.5) can be solved for the asset weights w in the following way:

$$
\begin{aligned}
& C * w = \lambda * r + \gamma * \mathbf{1} \\
\Updownarrow & \qquad\qquad (6.8) \\
& w = \lambda * C^{-1} * r + \gamma * C^{-1} * \mathbf{1}. \qquad (6.9)
\end{aligned}
$$

C will in general have full rank because the assets included in the investment universe will not be perfect substitutes (otherwise they would not be different assets to begin with) and for the same reason it will not be possible to form linear combinations of the assets that are identical. When C has full rank it is invertible and the above calculation is possible. Equation (6.9) shows the portfolio weights as a function of (C, r, λ, γ). The first two are direct inputs to the investment process, and although they may be very difficult to estimate to a sensible level of precision, for the purpose at hand they can be regarded as "observed" variables. What is not known are the values of λ and γ, and since these parameters follow from the way the optimisation problem is solved, we do not have any way of guessing their values or estimate them from other observed variables.[3] Instead their values have to be derived. One idea in this direction is to form two equations with two unknowns. Starting with (6.9) and premultiplying with r^T gives:

$$
\begin{aligned}
& r^T w = \lambda * r^T * C^{-1} * r + \gamma * r^T * C^{-1} * \mathbf{1} \\
\Updownarrow & \\
& r_p = \lambda * \left(r^T * C^{-1} * r\right) + \gamma * \left(r^T * C^{-1} * \mathbf{1}\right) \\
\Downarrow & \\
& r_p = \lambda * X + \gamma * Y.
\end{aligned}
$$

Taking again (6.9) and premultiplying with $\mathbf{1}^T$ gives:

[3] In economic optimisation theory λ and γ are referred to as shadow prices for the imposed constraints.

$$
\begin{aligned}
& \mathbf{1}^T w = \lambda * \mathbf{1}^T * C^{-1} * r + \gamma * \mathbf{1}^T * C^{-1} * \mathbf{1} \\
\Updownarrow & \\
& 1 = \lambda * \left(\mathbf{1}^T * C^{-1} * r\right) + \gamma * \left(\mathbf{1}^T * C^{-1} * \mathbf{1}\right) \\
\Updownarrow & \\
& 1 = \lambda * Y^T + \gamma * Z.
\end{aligned}
$$

To summarise, the two equations with two unknowns can then be written as:

$$
\left[\begin{array}{c} r_p \\ 1 \end{array} \right] = \left[\begin{array}{cc} X & Y \\ Y & Z \end{array} \right] \left[\begin{array}{c} \lambda \\ \gamma \end{array} \right],
$$

since it is noted that the entries in the 2-by-2 matrix on the RHS are all scalars so, $Y^T = Y$. Let's define:

$$
\begin{aligned}
X &= r^T * C^{-1} * r \\
Y &= r^T * C^{-1} * \mathbf{1} = \mathbf{1}^T * C^{-1} * r \\
Z &= \mathbf{1}^T * C^{-1} * \mathbf{1} \\
D &= X * Z - Y^2.
\end{aligned}
$$

This system of equations can be solved by Cramer's rule which states that:

$$
\lambda = \frac{\det\left(\left[\begin{array}{cc} r_p & Y \\ 1 & Z \end{array} \right]\right)}{\det\left(\left[\begin{array}{cc} X & Y \\ Y & Z \end{array} \right]\right)} = \frac{Z * r_p - Y}{D},
$$

and

$$
\gamma = \frac{\det\left(\left[\begin{array}{cc} X & r_p \\ Y & 1 \end{array} \right]\right)}{\det\left(\left[\begin{array}{cc} X & Y \\ Y & Z \end{array} \right]\right)} = \frac{X - Y * r_p}{D}.
$$

Based on this it is now possible to express the portfolio weights as functions of C and r alone. An expression for w can be found by inserting the expressions for λ and γ in expression (6.9):

$$
\begin{aligned}
w_p &= \lambda_p * C^{-1} * r + \gamma_p * C^{-1} * \mathbf{1} \\
&= \frac{Z * r_p - Y}{D} * C^{-1} * r + \frac{X - Y * r_p}{D} * C^{-1} * \mathbf{1} \\
&= \frac{1}{D}\left[X * C^{-1} * \mathbf{1} - Y * C^{-1} * r + Z * C^{-1} * r * r_p - Y * C^{-1} * \mathbf{1} * r_p\right] \\
&= \frac{1}{D}\left[\left(X * C^{-1} * \mathbf{1} - Y * C^{-1} * r\right) + \left(Z * C^{-1} * r - Y * C^{-1} * \mathbf{1}\right) * r_p\right] \\
&= g + h * r_p, \qquad (6.10)
\end{aligned}
$$

where:

$$
\begin{aligned}
g &= \frac{1}{D} * \left(X * C^{-1} * \mathbf{1} - Y * C^{-1} * r\right) \\
h &= \frac{1}{D} * \left(Z * C^{-1} * r - Y * C^{-1} * \mathbf{1}\right).
\end{aligned}
$$

In essence (6.10) shows how all unique and distinct frontier portfolios can be generated simply by varying the target return r_p. This is not a surprise since the same conclusion follows from the optimisation problem written in (6.3); however, the benefit of (6.10) is that it shows the solution directly and does not require numerical optimisation techniques to find a solution as it is the case for (6.3).

Let's see this used in practise. The following Matlab(R) script implements the above expression and derives the efficient frontier. To make sure that the calculations are correct we compare the derives efficient frontier using (6.10) to a "brute-force" calculation of the efficient frontier. The latter loops over a large set of possible combinations of asset weights, checks whether the allocation is feasible (i.e. whether the weights sum to unity) and then stores the standard deviation and expected return of all feasible portfolios. It should be clear that the "brute-force" calculation does not constitute a viable calculation scheme, especially when the number of assets grows.

```
[1] % --- Input parameters ---
[2] r = [ 3.25; 5.75; 7.5; 12.3 ];
[3] C = [ 25.0 24.5 33.0 37.5;
[4] 24.5 49.0 50.0 63.0;
```

```
[5] 33.0 50.0 121.0 66.0;
[6] 37.5 63.0 66.0 225.0 ];
[7] one = ones(4,1);
[8]% --- Expression for the weights ---
[9] X = r'*inv(C)*r;
[10] Y = r'*inv(C)*one;
[11] Z = one'*inv(C)*one;
[12] D = X*Z-Y^2;
[13] g = 1/D*(X*(inv(C)*one)-Y*(inv(C)*r));
[14] h = 1/D*(Z*(inv(C)*r)-Y*(inv(C)*one));
[15] r_max = 20;
[16] r_min = -5;
[17] incr = 0.05;
[18] zz = 1;
[19] for ( j=r_min:incr:r_max )
[20]    w_j = g+h*j;
[21]    W_p(zz,:)  = w_j';
[22]    R_p(zz,1) = w_j'*r;
[23]    S_p(zz,1) = sqrt(w_j'*C*w_j);
[24]    zz=zz+1;
[25] end
[26]% --- Brute Force calculation ---
[27] nObs = 10;
[28] zz = 0;
[29] for ( j=-1:0.1:1 )
[30]    for ( k=-1:0.1:1 )
[31]       for ( l=-1:0.1:1 )
[32]          for ( m=-1:0.1:1 )
[33]             w_bf = [ j;k;l;m ];
[34]             if ( sum(w_bf)==1 )
[35]                zz = zz+1;
[36]                W_bf(zz,:)  = w_bf';
[37]                R_bf(zz,1) = w_bf'*r;
[38]                S_bf(zz,1) = sqrt(w_bf'*C*w_bf);
[39]             end
[40]          end
[41]       end
[42]    end
```

```
[43] end
[44] figure
[45] plot(S_p,R_p,'k-','linewidth',2), ...
[46]           xlabel('Standard deviation'), ylabel('Expected return');
[47] hold on
[48] plot(S_bf,R_bf,'kx','linewidth',0.25);
```

The clear disadvantage of the brute force calculations as presented by lines [26]-[44] is that the number of for-loops increases by one for each additional asset that is included in the eligible investment universe; in addition it calculates some information that is not particularly useful namely all the interior points portfolios i.e. the portfolios that are not located on the efficient frontier. The implementation of the formula in (6.10) circumvents both of these drawbacks by only calculating frontier portfolios and as such only requires one loop running over the target return. This is illustrated in lines [19]-[25]. Lines [9]-[14] define the input variables facilitating the calculations as illustrated above and lines [15]-[17] define for which return targets the efficient frontier should be calculated and in which step size. Within each of the loops the calculation of the portfolio metrics is performed following the definitions in (6.1) and (6.2). Figure 6.2 illustrates the results generated by the script file. The "x"s in the figure represent the brute-force calculations and the bold line the results obtained when applying (6.10).

6.3 Diversification

As can be seen from the expression of the portfolio variance in equation (6.2) there are two terms to the calculation: one comes from the variance of the eligible investment assets i.e. from the diagonal of C, when $n = m$, the other from the covariance effects, i.e. from the off-diagonal elements of C, when $n \neq m$. Expression (6.2) can be rewritten to reflect this more clearly:

$$\sigma_p^2 = \underbrace{\sum_{n=1}^{N} w_n^2 * \sigma_n^2}_{\text{Variance effects}} + \underbrace{\sum_{n=1}^{N} \sum_{\substack{m=1 \\ m \neq n}}^{N} w_n * w_m * \sigma_{n,m}}_{\text{Covariance effects}} \tag{6.11}$$

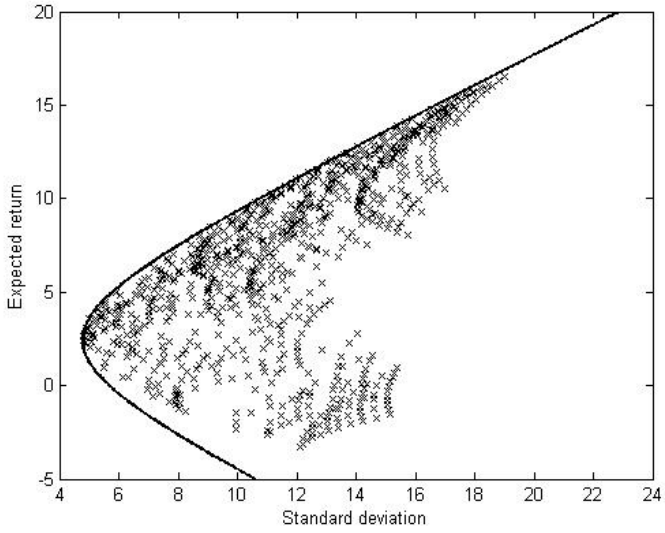


Figure 6.2: The possible investment frontier

To illustrate the effect of diversifying ones' capital among many assets, assume that $1/N$th of the investor's capital is invested in each of the N assets in the eligible investment universe. Using this assumption in (6.11) leads to:

$$\begin{aligned}
\sigma_p^2 &= \sum_{n=1}^{N}(1/N)^2 * \sigma_n^2 + \sum_{n=1}^{N}\sum_{\substack{m=1 \\ m\neq n}}^{N}(1/N) * (1/N) * \sigma_{n,m} \\
&= \frac{1}{N}\sum_{n=1}^{N}\frac{\sigma_j^2}{N} + \frac{N-1}{N}\sum_{n=1}^{N}\sum_{\substack{m=1 \\ m\neq n}}^{N}\frac{\sigma_{n,m}}{N*(N-1)} \\
&= \frac{1}{N}\overline{\sigma}_j^2 + \frac{N-1}{N}\overline{\sigma}_{n,m},
\end{aligned}$$

where the factorisation of $(1/N)$ outside the summation in the second row leaves the remaining part of the first term equalling the average variance. And, the factorisation of $(N-1)/N$ outside the double summation in the second term of the second row, leaves the remainder equal to the average

covariance.[4] Letting N approach infinity gives the following limits:

$$\lim_{N\to\infty} \frac{1}{N}\overline{\sigma}_j^2 \to 0$$
$$\lim_{N\to\infty} \frac{N-1}{N}\overline{\sigma}_{n,m} \to \overline{\sigma}_{n,m}.$$

In other words: when the investors' capital is well distributed among the eligible investment vehicles, the risk that stems from the individual assets is diversified away and what remains approaches the average covariance. Hence, the covariance terms, in this case the average covariance, cannot be diversified away. This part of the risk is also referred to as the systematic risk of the portfolio. In reality it is (naturally) not possible to distribute the capital between an infinite number of assets nor is it necessary for achieving diversification benefits. In most cases the diversification effect is considerable even when only fifteen assets are included in the portfolio and the marginal benefit of including more than fifty assets is for all practical purposes equal to zero.[5]

To illustrate the issue of diversification the Matlab(R) script below simulates a covariance matrix and calculates the variance of portfolios with an increasing number of assets included.

```
[1] nObs = 100;
[2] bgn = 2;
[3] xx = (bgn+1:1:nObs)';
[4] temp = tril(rand(nObs,nObs));
[5] temp = temp+temp';
[6] std = rand(nObs,1)+1;
[7] corr = temp-diag(diag(temp))+diag(ones(nObs,1));
[8] cov = (std*std').*corr;
[9] for ( j=1:nObs-bgn )
[10]    C = cov(1:j+bgn,1:j+bgn);
```

[4]Note that there are $N*(N-1)$ off-diagonal terms in the covariance matrix so dividing the sum of the covariances by this factor gives the average covariance.

[5]This statement is only true when dealing with an investment universe where there is one major risk factor, such as it is the case for equities and government bonds. This statement is for example not true when investigating an asset universe comprising corporate bonds.

```
[11]    w = 1/j.*ones(j+bgn,1);
[12]    sig2(j,1) = w'*C*w;
[13] end
[14] sig2pct = sig2./sig2(1,1).*100;
[15] figure
[16] plot(xx,sig2pct),xlabel('Number of assets in portfolio'),...
[17]                  ylabel('Risk %');
```

The output of the script is shown in Figure 6.3, which confirms the above mentioned diversification gains when including additional asset in the portfolio. It is worth remembering, though, that the Matlab(R) script uses a simulated covariance matrix which may not fully reflect covariance matrices estimated from real life data. Still, the example proves the point and shows how important it is to be aware of diversification benefits.

A description of the Matlab(R) script is given here: lines [1]-[3] define how many entries there should be in the covariance matrix, i.e. how many assets that are contained by the eligible investment universe, and that two assets should be included in the first calculation. the "xx" in line [3] is simply a counting variable used when plotting the result. Lines [4]-[9] simulate the covariance matrix. Line [4] generates the lower triangular matrix from a matrix of uniformly distributed random variables between zero and one. A covariance matrix is symmetric and therefore the lower triangular part of the matrix is taken so that a symmetric matrix can be generated in line [5]. Line [6] generates a vector of standard deviations, one for each asset; a lower bound of unity of the standard deviation is arbitrarily chosen. In line [7] it is ensured that the diagonal of the correlation matrix is equal to unity. Line [8] generates the covariance matrix following the calculation: $cov(x_1, x_2) = std(x_1) * std(x_2) * corr(x_1, x_2)$. This is done through element-by-element multiplication of the covariance and correlation matrices. Lines [9]-[13] calculate the portfolio variance for an investment universe that increases in size by one asset from one iteration to the next. The rest of the code plots the results.

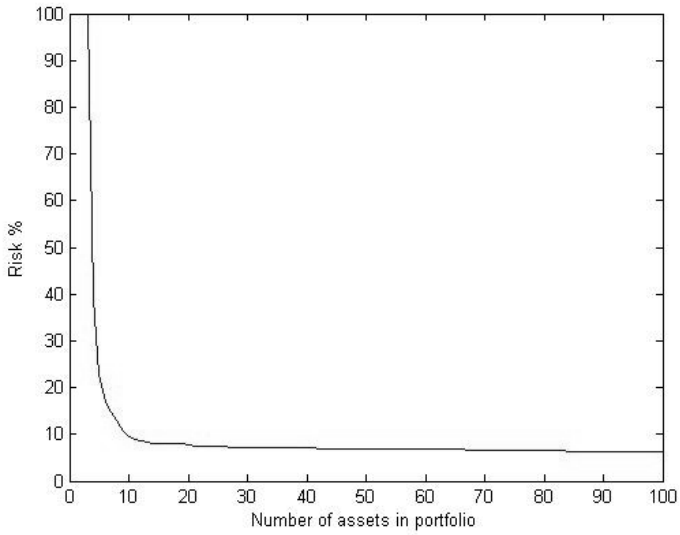


Figure 6.3: Example of the diversification effect

6.4 The minimum variance portfolio

It is recalled that the efficient frontier comprises the portfolios that have the lowest variance for a given level of return [see equation (6.3)]. A relevant question to ask is then: what are the weights for the global minimum variance portfolio? This portfolio is interesting because it tells us, based on a given investment universe, what the level of risk and return is for the least risky investment option, apart of course from the risk less asset, if such an asset exists.[6] Using the definition of the portfolio variance the global minimum risk portfolio can be identified:

[6]In asset pricing models the concept of a risk less asset is used as an anchor point, and often the return on a short term government bond / bank account is used as a proxy. However, it is worth emphasising that strong assumptions about the investment horizon are implied by the choice of such a risk free invest alternative. For example, if the investment horizon is one year (or longer) and a 1 month deposit rate is used as a proxy for the risk free rate, it is not really risk free because the deposit matures after one month and thus needs to be rolled over into a new deposit having a possibly different interest rate.

$$\begin{aligned}
\sigma_p^2 &= w_p^T * C * w_p \\
&= w_p^T * C * (\lambda * C^{-1} * r + \gamma * C^{-1} * \mathbf{1}) \\
&= \lambda * w_p^T * r + \gamma * w_p^T * \mathbf{1} \\
&= \lambda * r_p + \gamma \\
&= \frac{Z * r_p - Y}{D} * r_p + \frac{X - Y * r_p}{D} \\
&= \frac{1}{D}\left[Z * r_p^2 - 2 * Y * r_p + X\right] \qquad (6.12)
\end{aligned}$$

where the second line follows from (6.9), the fifth line substitutes in the expressions for λ and γ derived above, and the last line appears when reorganising terms. Clearly, (6.12) is a quadratic equation in r_p with the following minimum variance point:

$$\begin{aligned}
\left\{\sigma_{mvp}^2, r_{mvp}\right\} &= t_{1,2} = \left(\frac{4 * Z * X - 4 * Y^2}{4 * Z * (Z * X - Y^2)}, \frac{Y}{Z}\right) \\
&= \left(\frac{1}{Z}, \frac{Y}{Z}\right). \qquad (6.13)
\end{aligned}$$

Using this result to calculate the (x,y) coordinates for the minimum variance portfolio (mvp) in the example used to produce Figure 6.2 gives $\sigma_{mvp} = \sqrt{1/Z} = 4.78$; $r_{mvp} = 2.44$. To calculate the weights that define the mvp, (6.10) is used an the result for the return of the minimum variance portfolio is inserted:

$$\begin{aligned}
w_{mvp} &= g + h * r_{mvp} = g + h * \frac{Y}{Z} \\
&= \frac{C^{-1} * \mathbf{1}}{\mathbf{1}^T * C^{-1} * \mathbf{1}}
\end{aligned}$$

which can verified by using the symbolic calculation capabilities of Matlab(R):

```
[1] syms x y z r ic i f;
[2] f = 1/(x*z-y^2)*(x*ic*i-y*ic*r)+1/(x*z-y^2)*(z*ic*r-y*ic*i)*(y/z);
```

```
[3] pretty(simplify(f))
ans =
i*ic/z
```

Line [1] defines the symbolic variables that are needed. Line [2] defines f as the expression that should be simplified, and line [3] asks Matlab(R) to do the calculations. Since $Z = \mathbf{1}^T * C^{-1} * \mathbf{1}$, the weights for the minimum variance portfolio are given by the row sums of the inverse covariance matrix ($C^{-1} * \mathbf{1}$) divided by the row and column sums of the inverse covariance matrix. The latter is a scalar which makes sure that the calculated weights sum to unity. Let's confirm this calculation by using the four asset example from above:

```
[1] % --- Input values ---
[2] r = [ 3.25; 5.75; 7.5; 12.3 ];
[3] C = [ 25.0 24.5 33.0 37.5;
[4] 24.5 49.0 50.0 63.0;
[5] 33.0 50.0 121.0 66.0;
[6] 37.5 63.0 66.0 225.0 ];
[7] one = ones(4,1);
[8] % --- Doing the calculations ---
[9] w_mvp = inv(C)*one./(one'*inv(C)*one)
[10] sig_mvp = sqrt(w_mvp'*C*w_mvp)
[11] r_mvp = w_mvp'*r
w_mvp =
     1.0127
     0.1938
     -0.12032
     -0.086126
sig_mvp =
     4.7816
r_mvp =
     2.4437
```

Lines [1]-[7] determine the input values. Lines [9]-[11] implement the formulas; the results are $\sigma_{mvp} = 4.78$ and $r_{mvp} = 2.44$ exactly as above. The weights for the mvp are found to be $w_{mvp} = (1.0127,\ 0.1938,\ -0.1203,\ -0.0861)^T$.

6.5 Asset weight constraints

All of the above conclusions and derivations assume that there are no constraints on the asset weights and that it is possible to short sell assets, i.e. having negative weights in one or more assets covered by the eligible investment universe. Practical applications of portfolio optimisation will often impose constraints on given assets, either in the form of general maximum and/or minimum holding constraints on individual assets and/or asset groups. For example, a commonly imposed constraint is a "no-short sales constraint" for all assets. Once such constraints are permitted it is rather difficult to derive closed form expressions for the efficient frontier, if possible at all. In effect, the solution to the efficient frontier problem has to rely on numerical techniques.

A general form of the portfolio optimisation problem is stated in (6.14):

$$\begin{aligned} &\min\ \sigma_p^2 = w_p^T * C * w_p \\ &st. \\ &w_p^T * r = r_p \\ &w_p^T * \mathbf{1} = 1 \\ &lb \leq w_p \leq ub \\ &G * w_p \leq ubg \end{aligned} \tag{6.14}$$

Here the new constraints three and four allow for lower and upper bounds (lb and ub, respectively) on the individual portfolio weights, and for lower and upper bounds on groups of assets (lbg and ubg, respectively). The group definition is contained in the matrix G which has number of rows equal to the number of asset group definitions and columns equal to the number of assets.

Matlab(R) contains a portfolio optimisation module called $frontcon$ that sells with the Financial Toolbox. However, to illustrate how portfolio optimisation subject to constraints can be implemented, and to assist the unfortunate who do own the $FinancialToolbox$, below an alternative to $frontcon$ is constructed. This function is called $frontier$ and utilises the equations derived in Section 6.2 when no constraints are imposed on the optimisation problem, and a numerical solution to the objective function when constraints are imposed. To produce the numerical solutions to facilitate calculations in

the latter case, Matlab(R)'s optimiser *fmincon* is used along with the following calculation strategy:

a) Find the maximum return portfolio by solving:

$$\begin{aligned} &\max_{w} \; w^T * r \\ &st. \\ &w_p^T * \mathbf{1} = 1 \\ &lb \leq w_p \leq ub \\ &G * w_p \leq ubg \end{aligned} \tag{6.15}$$

b) Find the minimum risk portfolio by solving:

$$\begin{aligned} &\min_{w} \; w^T * C * w \\ &st. \\ &w_p^T * \mathbf{1} = 1 \\ &lb \leq w_p \leq ub \\ &g * w_p \leq ubg \end{aligned} \tag{6.16}$$

c) Find a desired number of efficient portfolios between the minimum risk and maximum return portfolios as the solution to (6.14) by varying the return requirement $w_p^T * r = r_p$ in appropriately defined steps.

This strategy is implemented in the Matlab(R) script below. Since the program is a bit long its annotation will be split into shorter parts. Note that no error-checks are implemented in the code. Hence it is not tested whether the dimensions of the input arguments match and whether all necessary inputs are provided. The implementation of an error-checking procedure is left as an exercise at the end of the chapter.

```
[1] function [out] = frontier(in)
[2]%
[3]% Chapter:  Asset Allocation
```

```
[4]%
[5]% Usage:
[6]%     [out] = frontier(in)
[7]%
[8]% in (structure):
[9]%      .C - covariance matrix r-by-r
[10]%     .R - vector of expected returns r-by-1
[11]%     .B - matrix of lower and upper bounds r-by-2
[12]%     .G - matrix of group definitions ngroups-by-r
[13]%     .GB - matrix of group bounds ngroups-by-2
[14]%     .N - number of points to be calculated on the frontier 1-by-1
[15]%
[16]% out (structure):
[17]%     .RR - matrix of risk and return obs on the efficient frontier N-by2
[18]%     .W - matrix of weights for the frontier points N-by-r
[19]%     .mvp - vector of stdandard dev and return (in that order) for
[20]%        ...the Min Var portfolio
[21]%     .mvpW - weights for global minimum var portfolio
[22]%
[23]% date:  October 2006
[24]% report bugs to:  email@kennyholm.com
[25]%
```

The first line uses the key word "function" that tells Matlab(R) that what follows should be taken as one entity, and that the first argument collected in hard brackets "out" is the return argument(s) of the function, which is set equal to the function name with input argument(s) i.e. "frontier" is in this case the function name, and "in" contains the input argument(s) to the function. Lines [2]-[24] all begin with "%" which means that they are not processed by Matlab(R). Hence, these lines are only comments meant to help the user. In fact, the first part of a function enclosed by "%" is what Matlab(R) will return when the user types "help functionname" at the command prompt. In our case, typing "help frontier" will display a print of the lines [2]-[24]. It is clear that structured variables are used as input and output arguments to and from the function. Lines [8]-[14] explain the input arguments while lines [16]-[21] delineate the output arguments. Lines [23]-[24] report the date the function was written and who to complain to when it doesn't work!

```
[26] warning off all
[27] % --- Organising inputs ---
[28] C=in.C; r=in.R; B=in.B; G=in.G; GB=in.GB; Np=in.N;
[29] nAssets = length(r);
[30] one = ones(nAssets,1);
[31] % --- Calculations for the global MVP ---
[32] w_mvp = inv(C)*one./(one'*inv(C)*one);
[33] sig_mvp = sqrt(w_mvp'*C*w_mvp);
[34] r_mvp = w_mvp'*r;
```

Line [26] turns off warnings. This is not strictly necessary but ensures that Matlab(R) doesn't print messages that may confuse the user of the function. Lines [27]-[30] organise the input variables, they so to speak unpack the structured variable containing the relevant input arguments and generate an auxiliary variable "one" that serves the role of the vector **1** used in Section 6.2. Lines [31]-[34] implement the expressions relevant for the minimum variance portfolio when no constraints are imposed.

```
[35] % Optimisation without constraints
[36] if ( isempty(B) & isempty(G) )
[37]    X = r'*inv(C)*r;
[38]    Y = r'*inv(C)*one;
[39]    Z = one'*inv(C)*one;
[40]    D = X*Z-Y^2;
[41]    g = 1/D*(X*(inv(C)*one)-Y*(inv(C)*r));
[42]    h = 1/D*(Z*(inv(C)*r)-Y*(inv(C)*one));
[43]    r_min = 0;
[44]    r_max = 2*max(r);
[45]    incr = (r_max-r_min)*1/(Np-1);
[46]    zz = 1;
[47]    for ( j=r_min:incr:r_max )
[48]       w_j = g+h*j;
[49]       W_p(zz,:)  = w_j';
[50]       S_p(zz,1) = sqrt(w_j'*C*w_j);
[51]       R_p(zz,1) = w_j'*r;
[52]       zz=zz+1;
[53]    end
```

The lines [35]-[52] implement the mathematics of the efficient frontier as derived in Section 6.2. Line [36] checks whether constraints are imposed by asking whether the matrix of asset bounds B and asset group definitions in G are empty. If this is the case the calculations in lines [37]-[52] are applied. Otherwise the lines below, following the solution strategy outlines in steps (a)-(c) above, are applied.

```
[54] else
[55]    options_ = optimset('LevenbergMarquardt','on','LargeScale','off','Display','off');
[56]    [W_max] = fmincon(@(w)-w'*r, w_mvp,G,GB,one',1,B(:,1),B(:,2),[],options_);
[57]    [W_min] = fmincon(@(w)w'*C*w, w_mvp,G,GB,one',1,B(:,1),B(:,2),[],options_);
[58]    r_min = W_min'*r;
[59]    r_max = W_max'*r;
[60]    incr = (r_max-r_min)*1/(Np-1);
[61]    zz = 1;
[62]    for ( j=r_min:incr:r_max )
[63]       [w_j] = fmincon(@(w)w'*C*w,w_mvp,G,GB,[one'; r'],[1; j],B(:,1),B(:,2),[],options_);
[64]       W_p(zz,:)  = w_j';
[65]       S_p(zz,1) = sqrt(w_j'*C*w_j);
[66]       R_p(zz,1) = w_j'*r;
[67]       zz=zz+1;
[68]    end
[69] end
```

The "else" in line [54] continues the "if" statement from line [36]. Hence, if constraints are imposed, and the calculations in lines [37]-[53] are not performed, then the calculations in lines [54]-[69] are to be performed. Line [55] specifies the "options" i.e. how the optimisation function $fmincon$ should be called.[7] Line [56] solves (6.15) and line [57] solves (6.16). In should be noted that the "-" in front of the objective function in line [56], i.e. "$-w^T * r$", implies that the function's maximum is found. $fmincon$ is designed to minimise functions, and "minus-minimisation" is equal to maximisation. Line [60] parcels the distance between the minimum and maximum attainable returns into the desired number of discrete points for which the efficient fron-

[7] For more information on this please investigate the help function of "optimset", hence type "help optimset" at the Matlab command prompt.

tier portfolios should be calculated as specified through the input parameter "in.N". Lines [62]-[68] perform the calculation of the efficient frontier portfolios following (6.14) for the different return targets defined by the for-loop structure in line [62]. And, line [69] concludes the if-statement initiated in line [36].

Finally, lines [70]-[73] collect the output in the structured variable: "out". Line [70] generates a matrix of the standard deviation and expected return of the frontier portfolios and line [71] collects the portfolio weights of the frontier portfolios. Lines [72]-[73] collect the output relevant for the global minimum variance portfolio i.e. the one derived without taking into account the portfolio constraints as shown in Section 6.4.

```
[70] out.RR = [S_p R_p];
[71] out.W = W_p;
[72] out.mvp = [sig_mvp r_mvp];
[73] out.mvpW = w_mvp;
```

The following example illustrates how the *frontier* function can be use. Assume that the eligible investment universe comprises four assets having the same expected returns and covariances as in the example on page 106, and assume further that short selling is prohibited i.e. all asset weights must be between 0 and 1, and that at least 25% of the portfolio should be allocated to assets $\{2, 3, 4\}$. The first constraint is taken into account by the asset weight bounds by defining

$$B = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

To ensure that at least 25% of the capital is allocated to assets $\{2, 3, 4\}$ we use the asset group constraint, which is formulated as (see, equation 6.14):

$$g * w_p \leq ubg.$$

In the current example we would probably prefer the constraint to read:

$$g * w_p \geq ubg,$$

which is the original constraint multiplied by -1. Hence, in the context of how the *frontier* function requires constraints to be specified, the desired constraint can be implemented in the following way:

$$g = \begin{bmatrix} 0 & -1 & -1 & -1 \end{bmatrix}, \text{ and } ubg = -0.25.$$

The example is shown below and compares the frontiers derived with and without the above mentions constraints.

```
[1] r = [ 3.25; 5.75; 7.5; 12.3 ];
[2] C = [ 25.0 24.5 33.0 37.5;
[3]           24.5 49.0 50.0 63.0;
[4]           33.0 50.0 121.0 66.0;
[5]           37.5 63.0 66.0 225.0 ];
[6] B = [zeros(4,1) ones(4,1)];
[7] G = [0 -1 -1 -1];
[8] GB = [-0.25];
[9] N = 30;
[10] % --- Without constraints ---
[11] in.R = r; in.C = C; in.B = []; in.G = []; in.GB = []; in.N = N;
[12] [out1] = frontier(in);
[13] % --- With constraints ---
[14] in.R = r; in.C = C; in.B = B; in.G = G; in.GB = GB; in.N = N;
[15] [out2] = frontier(in);
[16] % --- Plots ---
[17] figure
[18] plot(out1.RR(:,1),out1.RR(:,2),'k-','linewidth',1), xlabel('Risk'), ylabel('E[R]')
[19] hold on
[20] plot(out2.RR(:,1),out2.RR(:,2),'k:','linewidth',1), xlabel('Risk'), ylabel('E[R]')
[21] hold on
[22] plot(out1.mvp(1,1),out1.mvp(1,2),'ko','linewidth',1), xlabel('Risk'), ylabel('E[R]')
```

The following graph draws the comparison between the constrained and unconstrained efficient frontiers.

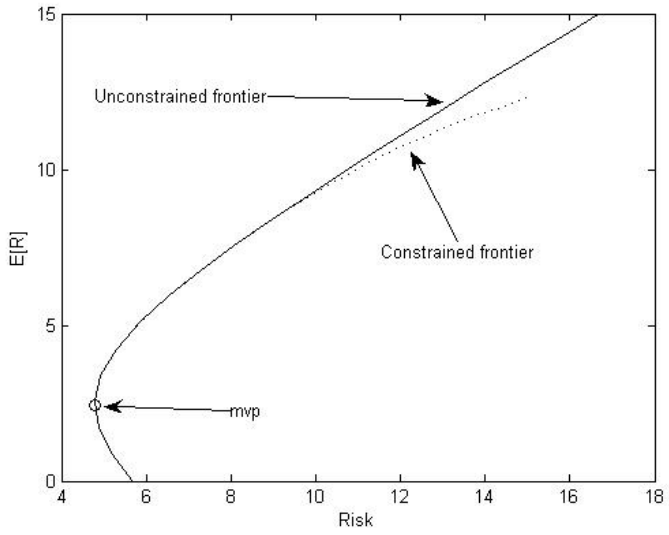


Figure 6.4: Example of an unconstrained and constrained efficient frontiers

6.6 The Capital Asset Pricing Model

A corner stone in modern financial theory is the answer to how equilibrium prices of financial contracts are determined in the economy. The Capital Asset Pricing Model (CAPM) provides one answer. In terms of a verbal argument the CAPM says that if the agents in the economy are only concerned with the two first moments of return distributions and have identical expectations to these moments i.e. they agree on the risk and return characteristics of the assets that trade in the economy, if they have similar investment horizons, if they are risk averse and prefer more return to less, if a risk free asset exists, and if agents in the economy can borrow and lend at the risk free rate, then all agents will hold a portfolio of two assets: one asset is the risk free interest rate, the other is the market portfolio. The agents will hold different portions of these two assets: risk averse agents will hold much more of the risk free asset and much less of the market portfolio than the less risk averse agents. The market portfolio is defined on the basis of the market capitalisation weights of the individual assets that trade in the economy. The market capitalisation weight for a given asset is defined as the ratio of the total value of that asset to the total value of all assets, i.e. $w_j = V_j / \sum_{k=1}^{K} V_k$, where j

refers to a specific asset and V to value and K to the total number of assets that trade in the economy. The value of an asset is equal to the number of "pieces" of that asset that trade in the economy, e.g. the number of stocks a given firm has issued or the number of bonds a given government has issued, multiplied by its price. Since the price enters the calculation of the value, and since agents in equilibrium all hold the market portfolio, albeit in different amounts to meet their risk aversion, the agent must have reached an agreement on the prices of the underlying assets, and hence the market is in equilibrium.

A graphical illustration of this is provided in Figure 6.5.

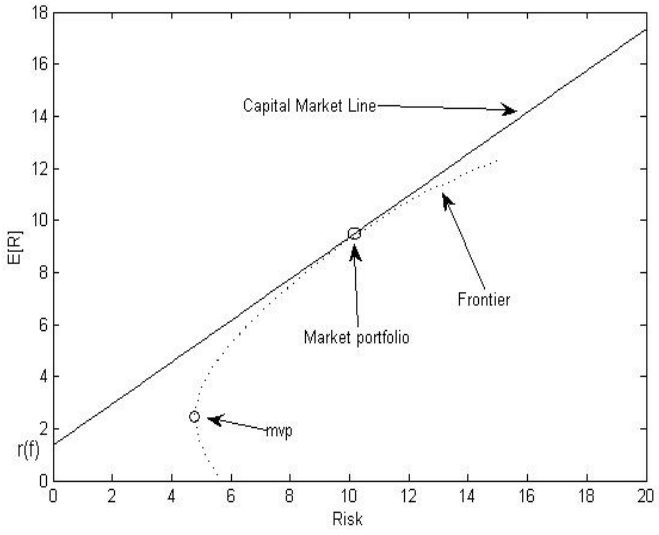


Figure 6.5: The Capital Market Line

The market portfolio is the tangent point between a straight line originating at the risk free rate and the portfolio frontier. This line is called "the capital market line" (cml). Its slope is equal to the market price of risk i.e. the compensation that agents require for bearing risk. In particular, the cml can formally be expressed as:

$$r_j = r_f + \frac{r_M - r_f}{\sigma_M}\sigma_j + e_{j.} \tag{6.17}$$

This is a linear relationship where σ_M and r_M are the standard deviation and expected return of the market portfolio, respectively, r_f is the risk free rate and σ_j is the standard deviation of portfolio j. The term e_j captures expectation errors, i.e. $e_j = E[r_j] - r_j$. Equation (6.17) in **not** the CAPM: it is not an equilibrium relationship since it only expresses a relationship for efficient portfolios, i.e. for the portfolios that fall on the cml. Recall that Section 6.3 showed that the standard deviation / variance of a portfolio decreases when more assets are added to it, and that the portfolio risk tended towards a certain constant in the limit defined by the average of the portfolio's covariance terms. The presence of a diversification effect suggests that the portfolio standard deviation is not a perfect measure of portfolio risk. Furthermore, since all investors in equilibrium hold only one risky portfolio, it is tempting to hypothesis that a better risk measure should somehow be related to this market portfolio. These ideas stem from the Capital Asset Pricing Model (CAPM) that was developed independently by Sharpe, Lintner and Mossin. It is intuitive to think about the CAPM as being a one factor model, i.e. a model that has only one underlying risky component. In the CAPM this risky component is the market portfolio. All other portfolios and individual assets can then be described by their co-movement with the single market factor. In particular, define this co-movement by:

$$\beta_j = \frac{cov(r_j, r_M)}{var(r_M)} = \frac{\sigma_{j,M}}{\sigma_M^2}. \tag{6.18}$$

Using this definition of risk the CAPM relation is written as:

$$r_j = r_f + \beta_j * (r_M - r_f) + e_j. \tag{6.19}$$

As it is the case for the cml in (6.17), the CAPM relationship in (6.19) is a linear relationship between return and risk also called the security market line (sml); however in (6.19) the risk is defined by β_j and not by σ_j. The former (β) is often referred to as the systematic risk of individual assets and portfolios, since this is the part of the risk that cannot be diversified away, while the latter (σ) is referred to the total risk of the portfolio/ the individual security. The variance of (6.19) is:

$$\underbrace{\sigma_j^2}_{\text{total risk}} = \underbrace{\beta_j^2 * \sigma_M^2}_{\text{systematic risk}} + \underbrace{\sigma_{e_j}^2}_{\text{unsystematic risk}}.$$

Financial markets are not going to reward agents for bearing risks they don't need to bear. Since the unsystematic risk can be diversified away, agents are not going to receive any compensation for having a portfolio that contains this risk source. The CAPM relationship reflects this basic principle.

Chapter 7

Statistical Tools

7.1 Introduction

This chapter presents some of the tools and techniques that can help modelling and projecting returns. As such, these tools can aid the process of estimating expected returns for fixed income securities, equities and currencies and help to model and forecast other variables of interest. The presentation is applied in nature: this means that the more mathematical oriented readers probably will be disappointed and feel compelled to complain about the lack of rigor. It is the hope, however, that the presentation will serve as a help to other readers, who may be interested in seeing actual implementation of the mentioned tools.

Learning objectives

- Master basic time-series models
- Understand the usefulness of regime-switching techniques and their applications
- Obtain additional insights into how yield curves can be modelled in the maturity and time-series dimensions

---

7.2 The Vector Auto Regression

The future evolution of most financial and economic time series depend on their own past evolution and cross correlation with past observations of other variables. Vector Autoregressive models exploit such dependency structures by characterising the observation at time t of a vector X as a function of past observations $\{X_{t-1}, X_{t-2}, \ldots, X_{t-j}\}$. The process for the mean of the vector series can be described by:

$$X_t = c + \sum_{j=1}^{p} X_{t-j} * A_j + e_t. \tag{7.1}$$

This is called a $VAR(p)$ model where p refers to the number of lags included in the model, and A_j collects the autoregressive parameters at lag j. In the formulation above, X_t is a matrix of dimension $(nObs - p) - by - nVars$, where $nVars$ refers to the number of variables contained in X.

The optimal lag-length can be determined by the use of summary statistics that are based on a specific weighting of the fit of the model against the number of parameters that are necessary to obtain this particular fit. Hence, these statistics gives a positive weight to the degree of fit obtained and includes a penalty for the number of variables needed to obtain the fit. In this way a parsimonious model can be found, i.e. one that uses the fewest possible variables to get the best relative fit. Parsimonious models have proven particularly useful for forecasting. Althought a model specification using many parameters often provides a very good in-sample fit, it is seldom the case that it also has the best best out-of-sample properties, i.e. produces the best forecasts. Rather the contrary is true, and it is therefore often useful to adherence to the principle of parsimony.

Two of such statistics are the Akaike's and Schwarz Information Criterions, labelled AIC and BIC, respectively. These can be calculated in the following way[1]:

$$AIC = \log\left(\widehat{\sigma}^2\right) + 2 * \frac{k}{nObs} \tag{7.2}$$

$$BIC = \log\left(\widehat{\sigma}^2\right) + k * \frac{\log(T)}{nObs} \tag{7.3}$$

[1] See e.g. Mills (1999)[pp.34-35]

It can be seen from (7.2) and (7.3) that the fit of the model is assessed by the log of the sum of sample squared residuals, $\widehat{\sigma}^2$, and then each criterion adds a specific penalty for the number of parameters used to obtain this fit: k represents the number of estimated parameters and $nObs$ is the number of time-series observations.

Related to (7.1) it is worth noting that any VAR(p) model can be written as a VAR(1) model; this is also called to write the VAR(p) model in companion form. Assume that we look at a VAR(3) model i.e.

$$X_t = c + X_{t-1} * A_1 + X_{t-2} * A_2 + X_{t-3} * A_3 + e_t.$$

By defining

$$\widetilde{X}_t = \begin{bmatrix} X_t \\ X_{t-1} \\ X_{t-2} \end{bmatrix}$$

and

$$\widetilde{A} = \begin{bmatrix} A_1 & A_2 & A_3 \\ I & 0 & 0 \\ 0 & I & 0 \end{bmatrix},$$

where I is the identity matrix of appropriate dimension, the VAR(3) can be written as a VAR(1):

$$\widetilde{X}_t = c + \widetilde{A} * \widetilde{X}_{t-1} + e_t. \tag{7.4}$$

For completeness it should be mentioned that the univariate version of the $VAR(p)$ model is called an autoregressive model (of order p), which is written as $AR(p)$:

$$x_t = c + a_1 * x_{t-1} + a_2 * x_{t-2} + \ldots + a_p * x_{t-p} + e_t.$$

7.2.1 Order of integration

A central concept in time series analysis is a classification of variables into their *order of integration*. Basically, this classification refers to how many times one needs to calculate first differences to make the time series stationary. Stationarity refers here to a visual picture where the time series

observations looks to be randomly scattered around a constant mean value.[2] It is naturally possible to test for whether a time series is stationary e.g. using the Dickey-Fuller test statistic; and one can rely on economic theory in judging whether a given time series is stationary or not.

The integration orders typically seen in finance and economics are of orders 0 and 1, written as $I(0)$ and $I(1)$, respectively. One rarely encounters time series of order 2 or higher. $I(0)$ refers to a stationary process and $I(1)$ to a process that is integrated of order 1. Hence, if Y_t is $I(1)$, then the process $\Delta Y_t = Y_t - Y_{t-1}$ will be $I(0)$. A standard example of an I(1) process is:

$$x_t = k + a * x_{t-1} + e_t,$$

with $a = 1$. This is the well-known random walk model. Stationarity in the univariate case is defined by the value of a. When $-1 < a < 1$, the process above is $I(0)$.

For a process involving more lags than one it is a bit more complicated to determine whether a process is stationary or not. The requirements are listed here, for detail see Hamilton (1994): For an AR(p) model it is required that the roots of the following polynomia all lie outside the unit circle:

$$1 - a_1 * z - a_2 * z^2 - \ldots - a_p * z^p = 0.$$

Similarly, for the VAR(p) model it is required that the roots of the following polynomia all lie outside the unit circle:

$$\det\left[I - A_1 * z - A_2 * z^2 - \ldots - A_p * z^p\right] = 0,$$

where **det** indicates the determinant. Given that a process is stationary its mean is calculated by:

$$E\left[X\right] = \frac{c}{I - \sum_{j=1}^{p} A_j},$$

for the $VAR(p)$ model and by:

$$E\left[x\right] = \frac{c}{1 - \sum_{j=1}^{p} a_j}$$

[2]The concept of *trend stationarity* refers to the situation where the time series observations looks to be randomly scattered around a constantly upwardly or downwardly trending line.

for the $AR(p)$ model.

It can be shown that the VAR(p) model can be estimated efficiently by the use of consecutive OLS regressions. Alternatively one can choose to maximise the value of the multivariate log-likelihood function assuming that errors are normally distributed. This amounts to calculating the approximative log likelihood of the multivariate normal distribution function, as given by:

$$\log L = \frac{nObs - p}{2} * \log\left[\det\left(\Omega^{-1}\right)\right] - \frac{1}{2} * \sum_{j=p+1}^{nObs} e_t * \Omega^{-1} * e_t^T.$$

In terms of speed however, since closed-form expressions exist for the solution to the OLS regression, this is typically faster to apply the OLS principle than it is to optimise the log likelihood function.

7.3 Regime switching models

7.3.1 Introduction

The intuition behind regime-switching models is perhaps best illustrated via a standard linear dummy-variable regression of the form:

$$x_t = c_1 d_{1,t} + c_2 d_{2,t} + c_3 d_{3,t} + e_t. \tag{7.5}$$

Here x is the data thought to contain two or more states, where s indicates the number of regimes, e.g. $s = 3$ means that three states are modelled, and the c's represent constants that define the mean in each state. The d variables represent dummy variables defining when a given state is effective. It is further assumed that the error term is normally distributed, $e \sim N(0, \sigma^2)$. To illustrate how data can look when the underlying data generating process contains regime-switches Figure 7.1 shows 300 realisations of a simulation of (7.5), where the process for d is chosen arbitrarily.

It is observed how the data is evolving around a fixed mean for some time, until an abrupt change is observed after which data evolves around a new mean for some time. These "steps" observed in the data represent the regimes. In Figure 7.1 we observe three regimes and my old eyes suggest that the means of the regimes are roughly $c_1 = 8$, $c_2 = 4$, and $c_3 = -2$. Below we will fit a regime-switching model to the data and find these mean estimated more accurately.

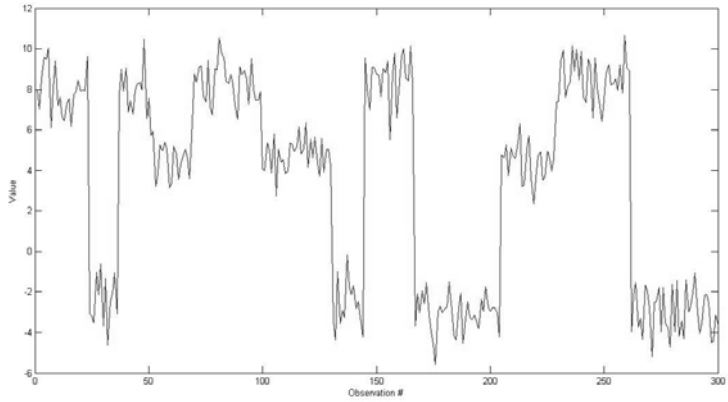


Figure 7.1: Example of regime-switching data

In a linear regression model as e.g. (7.5) usually the dummy variables are fixed exogenously, however, a regime-switching model in the above guise of equation (7.5), the d's as well as the c's are inferred simultaneously from the data. In order to facilitate estimation of the model, where the d's are going to be treated as unobservable, a prediction-updating algorithm is applied to construct the likelihood function.

To set up this algorithm a number of inputs need to be defined. These are: starting values for the parameters to be estimated, which in our example is: $\theta = \{c_1, c_2, c_3, \sigma^2, P\}$, where a transition probability matrix P gives the conditional probability of migrating within and between states. For example, a model comprising three states P would be:

$$P = \begin{bmatrix} p_{11} & p_{12} & 1 - p_{33} - p_{23} \\ p_{21} & p_{22} & p_{23} \\ 1 - p_{11} - p_{21} & 1 - p_{22} - p_{12} & p_{33} \end{bmatrix}, \tag{7.6}$$

where the column number corresponds the state which is active at time t and the row to the state which is active at $t + 1$, for example, the second column and third row entry (p_{23}) gives the probability that state 3 is followed by state 2. In general, the entries of P, i.e. $p_{(j,k)}$, indicate the transition probabilities of going from state k, at time t, to state j, at time $t + 1$. Consequently, columns must sum to unity, since each column expresses the probability the one state is followed by any other state. The diagonal of P expresses the

probability that the process stays in the same state and can be taken as an indication of the persistence of each state. Within the updating-prediction algorithm used to estimate regime-switching models it is assumed that state probabilities follow a markov chain with P as transition matrix.

State probabilities should be distinguished from the transition probabilities. The matrix P contains transition probabilities, i.e. the probabilities of migrating from one state to the next, when being in a given state at time t, and is of dimension $s - by - s$. In contrast, the vector of state probabilities gives the probability that the datapoint observed at time t is generated by either of the s regimes. These probabilities are collected in $\pi_{t|t}$, which is of dimension $s - by - 1$. For example, in the three state case, it could be that

$$\pi_{t|t} = \begin{bmatrix} 0.20 \\ 0.70 \\ 0.10 \end{bmatrix},$$

which would mean that there is 20% probability that the data point observed at time t is generated by state 1, 70% probability that it is generated by state 2 and 10% probability that it is generated by state 3. The slightly peculiar subscript comes from the prediction-updating algorithm described below. The matrix of values of the density for each observation, within each state, is collected in $D = f(x_t | s_t = j; \theta)$ for all observations t and states j. Here $f(\cdot)$ is taken to be the normal density, hence, again using the three state case as an example,

$$D_t = \begin{bmatrix} \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left\{\frac{-(x_t - c_1)^2}{2\sigma_1^2}\right\} \\ \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left\{\frac{-(x_t - c_2)^2}{2\sigma_2^2}\right\} \\ \frac{1}{\sqrt{2\pi}\sigma_3} \exp\left\{\frac{-(x_t - c_3)^2}{2\sigma_3^2}\right\} \end{bmatrix}. \tag{7.7}$$

As can be seen from (7.7) it is possible that both the mean and the variance of the data generating process varies over time. For the example data shown in Figure 7.1 it seems only to be the case that the the mean varies between regimes. However, in other applications regime-switches can just as well occur in the error term variances of the model. A regime dependent variance could for example comply with the notion that financial markets in periods of stress exhibit increased return volatility, compared to normal market circumstances. An appropriate model that could capture such behaviour would

be a two state model where the variance of the data generating process varies across states.

To estimate regime-switching models, Hamilton (see Hamilton (1994, Chapter 22)) devise a method whereby one iterates over

$$\pi_{t|t} = \frac{\pi_{t|t-1} \odot D_t}{\sum_{j=1}^{s} \pi_{t|t-1} \odot D_t}, \tag{7.8a}$$

$$\pi_{t+1|t} = P\pi_{t|t}\,, \tag{7.8b}$$

for all $t = 1, 2, \ldots T$. The sign $\odot$ indicates element-by-element multiplication.

The intuition of expression (7.8a) for $\pi_{t|t}$ consists of two parts. First, remember that $\pi_{t|t}$ gives the state probabilities at a given point in time. Intuitively, the better a data point observed at time t fit in either of the densities, as evaluated by inserting the data point into (7.7), the higher should this probability be. Second, given that we know the state probabilities at time $t-1$ and the transition matrix, then we can generate a projection for the state probabilities at time t using only information available at time $t-1$. The numerator of (7.8a) represents the combination of these two sources of information to draw inference about the which state that generates a given data point, by taking the element-by-element product of the fit at time t, as represented by the density observations collected in D_t, and the forecast of the state probability made for time t using information available at time $t-1$. The numerator is therefore of dimension $s - by - 1$. The denominator is the sum of the numerator-vector and can be seen to serve the purpose of normalising $\pi_{t|t}$ so that each entry in this vector falls on the interval $\{0, 1\}$, and can thus be interpreted as probabilities.

It so happens, that the denominator also is identical to the likelihood function value, conditional on θ.The estimation is then completed by optimising,

$$\log L(\theta) = \sum_{t=1}^{T} \log \left\{ \sum_{j=1}^{s} \pi_{t|t-1} \odot D_t \right\}. \tag{7.9}$$

In practice it is not always easy to find the number of regime-switches that are contained in data. Naturally, one needs to plot data before modelling it, and this can be used as a first indication for the choice on the number

of states to be included. Another, equally valid option, is to be guided by economic theory. Statistical tests have been devised to help determining the number of states present in data, however it is beyond the scope of the current text to venture into a description of these.

An example of the estimation of a univariate regime-switching model with three states is provided below using the data displayed in Figure 7.1. The presented code example is general in the sense that it can handle the estimation of univariate regime switching models with up to four states. The following functions are needed:

1) The main function that sets up the problem and calls the Matlab(R) optimisation module. This function is called *regime.m*[3]

2) The likelihood function is calculated by *regime_likeli.m*. This function calls two other functions: *regime_normaldensity* and *regime_pmat*.

2a) *regime_normaldensity* simply implements the density function of the normal distribution.

2b) *regime_pmat* sets up the transition matrix shown in (**??**) so that it matches the desired number of states to be estimated.

The below represents the main function of the regime-switching function called *regime.m*.

```
[1] function [ out_ ] = regime(dat,m,ml,mu,maxiter)
[2]% Estimate Univariate Regime Switching Model on data (dat)
[3]%
[4]% dat      - data to be used in the estimation
[5]% m        - starting values for the mean
[6]%                number of states equal to nobs in m are estimated (max=4)
[7]% ml       - lower bounds for estimated parameters
[8]% mu       - upper bounds for estimated parameters
[9]% maxiter  - maximum number of iterations
[10]%
[11]% ...checking and setting up data
```

[3]Naturally, regime-switching models can be more elaborate than what this function can hanlde. For example, autoregressive terms, regime-switching variances and time dependent transition probabilities could be included.

```
[12] warning off all;
[13] dat    = dat(:,1);       % ensuring that data is univariate
[14] S     = length(m(:));    % number of states
[15] sig    = std(dat)/5;     % starting value for the standard dev
[16] P_diag = 0.95;
[17] [T,junk] = size(dat);
[18] P        = ones(S,1).*P_diag;
[19]%.....................................
[20]% parameter constraints on transition matrices
[21]%.....................................
[22] if ( S==2 )
[23]    A = [];
[24]    A_l = [];
[25]    A_u = [];
[26] end
[27] if ( S==3 )
[28]    A = [ 0 0 0 0 1 0 0 1 0 0 ;
[29]          0 0 0 0 0 1 0 0 1 0 ;
[30]          0 0 0 0 0 0 1 0 0 1 ];
[31]    A_l = [ 0; 0; 0 ];
[32]    A_u = [ 1; 1; 1 ];
[33] end
[34] if ( S==4 )
[35]    A = [ 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 ;
[36]          0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 ;
[37]          0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 ;
[38]          0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 ];
[39]    A_l = [ 0; 0; 0; 0 ];
[40]    A_u = [ 1; 1; 1; 1 ];
[41] end
[42]% ...  starting values and structure for 'Prob'
[43] S_param = [ m; sig; P];
[44] if ( S==3 )
[45]    S_param = [ m; sig; P; 0.02; 0.02; 0.02];
[46] end
[47] if ( S==4 )
[48]    S_param = [ m; sig; P; 0.01; 0.01; 0.01; 0.01; 0.01; 0.01; 0.01; 0.01];
[49] end
```

```
[50] nparams = length(S_param);
[51] lb = -inf.*ones(nparams,1);
[52] ub = inf.*ones(nparams,1);
[53] lb(1:S,1) = ml;
[54] ub(1:S,1) = mu;
[55] if ( S==2 )
[56]    lb(nparams-S+1:nparams,1) = zeros(S,1);
[57]    ub(nparams-S+1:nparams,1) = ones(S,1);
[58] end
[59] if ( S==3 )
[60]    lb(nparams-(S*2)+1:nparams,1) = zeros(S*2,1);
[61]    ub(nparams-(S*2)+1:nparams,1) = ones(S*2,1);
[62] end
[63] if ( S==4 )
[64]    lb(nparams-(S*3)+1:nparams,1) = zeros(S*3,1);
[65]    ub(nparams-(S*3)+1:nparams,1) = ones(S*3,1);
[66] end
[67]% ...  structure for 'Prob.user'
[68] Prob.user.dat = dat;
[69] Prob.user.S   = S;
[70] Prob.user.T   = T;
[71]% ...  calling optimisation module
[72] options = optimset('Display','iter','LargeScale','off', ...
[73]                          'LevenbergMarquardt', 'on', 'HessUpdate', ...
[74]                          'steepdesc', 'MaxFunEvals', maxiter, 'TolFun', 1e-8);
[75] [param, fval, exitflag, outpt, la, g, H] = fmincon('regime_likeli', ...
[76]                          S_param,A,A_u,[],[],lb,ub,[],options,Prob);
[77] [ lik, Pr ] = regime_likeli(param,Prob);
[78] Pr = Pr(2:length(Pr),:);
[79] se = sqrt(diag(pinv(H)));
[80] p_out = zeros(length(Pr),2);
[81] p_out(1:length(param),:)  = [param se];
[82] out_ = [Pr p_out];
```

Line [1] specifies the name of the function to the "*regime*" and outlines the input arguments necessary to run the function. These are, *dat*, containing the data; m, the starting values for the mean parameters; ml, lower bounds for the estimated parameters; mu, upper bounds for the parameter

estimates; and, *maxiter*, setting an upper limit for the number of iterations that Matlab(R) is allowed to use. *maxiter* can be useful in an explorative analysis where the number of states and starting values are difficult to set a priori. In such cases this input parameter allows the user to limit calculation time by setting a relative low iteration limit, e.g. $maxiter = 100$. Line [12] turns off warnings, so the user is not bothered with messages intended for the program developer. Lines [13]-[18] fix necessary auxiliary variables and set starting values for some of the parameters to be estimated. For example, line [18] sets starting values for the transition matrix P (see, **??**) ensuring that it has the right dimension as determined by the variable S that contains the number of states to be estimates. S is allocated a value in line [14] on the basis of the dimension of the input parameter m. Lines [19]-[41] impose constraints so that the columns of P sum to unity, as they should. These constraints are implemented by calculating one column entry, in each column, residually. This is implemented by the A matrix with corresponding upper bounds as defined by A_u.[4] Such constraints can be included when using Matlab(R)'s built in optimiser: $fmincon$. It is seen that the dimension of the A matrix together with the dimensions for the upper and lower bounds are matched to the number of estimated states. This matching is done through the series of if statements. Specifically, this constraint is of the form:

$$A_{(n,n)} * x_{(n,1)} \leq A\_u_{(n,1)},$$

where x refers to the parameters to be estimated. In effect, A selects the parameters that are comprised by the constrains and sums them (because the matrix only contains entries equal to either 0 or 1) and A_u gives their upper limits. Lines [42]-[49] specify the starting values for the parameters of the model, collected by the vectors names "S_param". The standard upper and lower bounds for the parameters to be estimated are determined in lines [50]-[66] also using an if-structure to accommodate the user specified number of regimes. Data are allocated to the structured variable denoted by $Prob.user$ in lines [67]-[70]. Lines [72]-[74] define options for who the optimisations are to be carried out, and lines [75]-[76] call Matlab(R)'s built in optimiser fmincon to conduct the maximisation of the likelihood function of the model. The left-hand-side of the function call in line [75] specifies the output variables and

[4]It is noted that A_l is not used for anything. This constraint is not necessary because of the general parameter lower limits defined by the parameter *lb*.

the right-hand-side, starting with $fmincon('regime\_likeli'...$ tells Matlab(R) which function to optimise, in this case it is the function called *regime_likeli*. Line [77] makes an additional call to the function *regime_likeli* in order to facilitate the calculation of the state probabilities, collected in the matrix Pr, using the optimised values for the parameters. Alternatively, this matrix of state probabilities could be added to the structured variable *Prob.user* at the end of the *regime_likeli* function. The concluding lines of the regime function prepares the output and calculates standard errors for the parameter estimates.

The function *regime_likeli.m* implements the formulas shown in the text above to calculate the likelihood function of the regime switching model. In particular, equations (7.7), (7.8a), (7.8b), and (7.9) are used.

```
[1] function [ lik, Pr ] = regime_likeli(S_param,Prob)
[2]%
[3]% Likelihood function of the regime switching model
[4]% with switches in the mean
[5]% ...  unpacking data
[6] dat = Prob.user.dat;
[7] S   = Prob.user.S;
[8] T   = Prob.user.T;
[9]% ...parameters
[10] mean_   = S_param(1:S,1);
[11] sig_    = S_param(S+1,1);
[12] P_      = regime_pmat(S, S_param);
[13] Pr      = (1/S).*ones(S,T);      % Vector of state probabilities on each obs
[14] Cast_Pr = (1/S).*ones(S,T);      % Vector of forecasts on Pr(i+1) on each obs
[15] smooth  = Cast_Pr;               % Vector of smoothed probabilities
[16] N       = zeros(T,S);            % Matrix of densities
[17] lik     = zeros(T,1);            % Vector of likelihood values
[18] for ( j=1:S )
[19]    N(:,j) = regime_normaldensity( mean_(j,1),sig_,dat );
[20] end
[21] N = N';
[22] for ( i=2:T+1 )
[23]     lik(i) = ones(1,S)*( Cast_Pr(:,i-1).*N(:,i-1) );
[24]    Pr(:,i) = ( Cast_Pr(:,i-1).*N(:,i-1) )./lik(i);
[25]    nan_test = isnan(Pr);        %errortrap on the regime probabilities
```

```
[26]    nan_test = sum(nan_test(:)); %starting values are used if problems occur
[27]    if ( nan_test > 0 );
[28]       Pr(:,i) = (1/S).*ones(S,1);
[29]       lik(i) = 2.7;
[30]    end
[31]    Cast_Pr(:,i) = P_*Pr(:,i);
[32]    for ( c=1:S )
[33]       if ( Cast_Pr(c,i) == 0 );
[34]          Cast_Pr(c,i) = 0.001;
[35]       end;
[36]       if ( Pr(c,i)>1 );
[37]          Pr(c,i) = 1;
[38]       end;
[39]       if ( Pr(c,i) <0 );
[40]          Pr(c,i) = 0;
[41]       end;
[42]       c = c + 1;
[43]     end;
[44]     i = i+1;
[45] end;
[46] [row_lik,ol] = size(lik);
[47] lik = log(lik(2:row_lik-1,1));
[48] lik = -1*sum(lik);
[49] Pr = Pr';
```

Lines [2]-[17] prepare the input data and initiate the vectors and matrices necessary for calculating the likelihood function. Lines [18]-[20] make successive calls to the function calculating the normal density (this function is shown below: *regime_normaldensity*). The loop variable j counts the number of user selected regimes to be estimated and the *for loop* in lines [18]-[20] then calculates the normal density fit for the data given each of the means for the regimes to be estimated i.e. it implements equation (7.7). Line [22] introduced a *for loop* that ends in line [45]. This structure loops over the number of observations contained in the data sample and implements equation (7.8a) in line [24] and equation (7.8b) in line [31]. The likelihood function is calculated in line [23] and the log-likelihood values in lines [47]-[48]. The rest of the code relates to error-checks that ensure that the code works even if has poor starting values or in the case of numerical mishaps. For example, lines

[25]-[30] implements a safety valve against cases where the state probabilities cannot be calculated. In such cases the state probabilities are set equal to one divided by the number of states; Lines [32]-[45] takes care of abnormal events where the probabilities are calculated to be either greater than one or lower then zero. This naturally does not happen in normal applications of the model, however, it is advisable to implement such error checks to ensure that the code is resilient toward misguided input data and the like.

The function *regime_normaldensity*

```
[1] function [lik] = regime_normaldensity(mean_,sigma_,dat)
[2]%
[3]% normal density with mean1 and variance sigma1^2
[4]% data is dat
[5]%
[6] eps = dat - mean_;
[7] lik = inv(sqrt(2*pi*sigma_^2))*exp(-(eps .*eps)./(2*sigma_^2));
```

The function *regime_pmat* implements equation (**??**) ensuring that the dimension of the matrix corresponds to the number of states specified by the user. It is noted that the function *regime_pmat* is called by *regime_likeli* in line [12].

```
[1] function [ P_mat ] = regime_pmat(S, S_param)
[2]%
[3]% transition matrices depending on the number of states
[4]%
[5] nparams = length( S_param(:)  );
[6] nobs_p = nparams-S+1:nparams;
[7]
[8] if ( S==2 )
[9]    start_ = nparams-S+1;
[10]   p11 = S_param(start_,1);
[11]   p22 = S_param(start_+1,1);
[12]   P_mat = [ p11 1-p22;
[13]            1-p11 p22 ];
[14] end
[15] if ( S==3 )
[16]    start_ = nparams-(S*2)+1;
```

```
[17]    p11 = S_param(start_,1);
[18]    p22 = S_param(start_+1,1);
[19]    p33 = S_param(start_+2,1);
[20]    p21 = S_param(start_+3,1);
[21]    p12 = S_param(start_+4,1);
[22]    p23 = S_param(start_+5,1);
[23]    P_mat = [ p11 p12 1-p23-p33 ;
[24]              p21 p22 p23 ;
[25]         1-p11-p21 1-p12-p22 p33];
[26] end
[27] if ( S==4 )
[28]    start_ = nparams-(S*3)+1;
[29]    p11 = S_param(start_,1);
[30]    p22 = S_param(start_+1,1);
[31]    p33 = S_param(start_+2,1);
[32]    p44 = S_param(start_+3,1);
[33]    p21 = S_param(start_+4,1);
[34]    p32 = S_param(start_+5,1);
[35]    p12 = S_param(start_+6,1);
[36]    p23 = S_param(start_+7,1);
[37]    p34 = S_param(start_+8,1);
[38]    p31 = S_param(start_+9,1);
[39]    p13 = S_param(start_+10,1);
[40]    p24 = S_param(start_+11,1);
[41]    P_mat = [ p11 p12 p13 1-p24-p34-p44 ;
[42]              p21 p22 p23 p24 ;
[43]              p31 p32 p33 p34 ;
[44]         1-p11-p21-p31 1-p12-p22-p32 1-p13-p23-p33 p44 ];
[45] end
```

We now invoke the main regime-switching function (*regime.m*) to fit a three state model to the example data from Figure 7.1. This is done in Matlab(R) command line mode, where it is required that the data vector X lives in Matlab(R)'s workspace.

```
[1] [ RS_out ] = regime(X,[8;4;-2],[-10;-10;-10],[10;10;10],1000)
```

The results are contained in the structure *RS_out*. The estimated parameters are contained in RS_out.est and are shown below. The means of the regimes and the variance of the error term are estimated to be

$$\begin{aligned}\widehat{c}_1 &= 8.3269\\ \widehat{c}_2 &= 4.6129\\ \widehat{c}_3 &= -2.9086\\ \widehat{\sigma} &= 1.0214,\end{aligned}$$

and, the transition matrix is estimated to be

$$\widehat{P} = \begin{array}{ccc} 0.9590 & \textit{0.0275} & \textit{0.0196} \\ 0.0165 & 0.9589 & 0.0098 \\ \textit{0.0245} & 0.0136 & 0.9706 \end{array},$$

where numbers non-italics numbers represent the parameter estimates and the numbers in italics are calculated as the residual, since the columns sum to unity. In the plot below we super impose the estimated regime classifications for state 1, contained in $RS\_out.pr(:,1)$, onto the plot of the data. This allows us to visually inspect how well the regime-switching model classifies data into the hypothesised regime one. Plots can naturally be generated for the remaining two states to asses their relative fits. Such plots will show that the model fit the other regimes equally well, as also indicated by the parameter estimates.

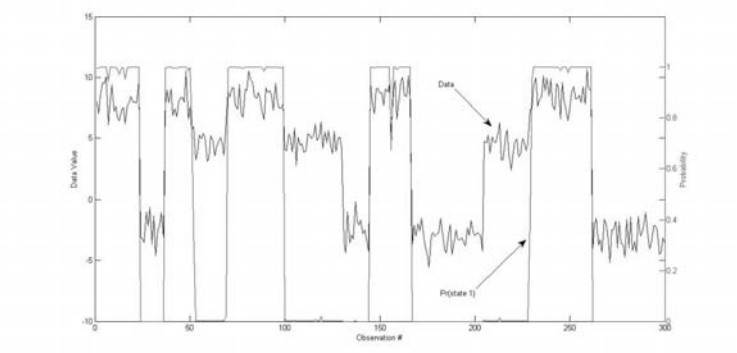


Figure 7.2: Data and estimated probability for state 1

7.4 Yield curve models in state-space form

This section introduces the state space form as a means to estimated time-series models. To help intuition along on this topic the framework is introduced by using the Nelson-Siegel model as an example. In addition, a general framework is presented allowing for regime-switches in the evolution of underlying factors, however, the framework naturally also apply to time series that do not exhibit regime switching behaviour.[5] The key ingredients to the state space model is the observation equation, the state equation and the filtering algorithm. The observation equation is a description of the phenomenon that is being investigated. In the case of the Nelson-Siegel model this is the yield curve equation. The state equation is a description of how the underlying factors evolve over time; these factors are taken to be unobserved and will be estimated using the kalman filter, which constitutes the third and last components to the framework.

The following represent the Nelson-Siegel model written in state-space form. First the observation equation:

$$Y_t(\tau) = H * \beta_t + e_t$$

and then the state equation:

$$\beta_t = k + F * \beta_{t-1} + v_t,$$

which without loss of generality can be written as a $VAR(1)$ model due to (7.4). Below a more detailed description of this framework is given.

7.4.1 The Nelson-Siegel model in state-space

To illustrate the inner working of the state-space modelling framework, the Kalman filter and the Hamilton regime-switching filter, the historical evolution of observable yield curves in the cross-sectional as well as the times series dimensions is modelled by the aid of the Nelson-Siegel model. The Nelson-Siegel model is a parametric description of yield curves that capture the cross-sectional dimension i.e. yields as a function of maturity. In the time series dimension it is assumed that the evolution of the Nelson-Siegel factors follow a regime-switching VAR(1) specification. The advantage of

[5] An excellent description of state space models with and without regime switches is given in Kim and Nelson (2000).

the suggested approach is that it, through the regime-switching component, identifies generic shapes of the yield curve as they have materialised in the past. In the context of strategic investment decisions, such generic curves facilitate easy generation of future yield evolutions, especially if the occurrence of the yield curve regimes are linked to macro economic variables.

The model presented here is set in state space form and can be seen as an expansion of the approach taken by Diebold and Li (2006). In particular, we include the possibility of regime switches in the state equation accounting for recurring structural breaks in the time-series dimension of the underlying yield curve factors.

Adding regime switches

The parsimonious representation of yield curves as suggested by Nelson and Siegel (1987) lends itself to an intuitive interpretation. It uses three factors to describe the yield curve at a particular point in time, these being: the level (β_1), which is the yield at infinite maturity, the slope (β_2), which is the difference between the long and the short end of the curve, and the curvature (β_3), which indicates how much the curve 'bends' up or down, as outlined in Section 5.2.1. To facilitate estimation of structural changes in the yield curve factors the state equation allows for regime switches in the mean parameters. The economic and practical motivations for this is summarised by the following points:

- The yield is a function of the underlying economic environment. It is therefore reasonable to believe that yield curves have different shapes contingent on the prevalent economic environment. In particular, during economic expansion the yield curve is generally very steep, whereas it during inflationary periods tends to be flat or inverse. A justification for these generic shapes can be found in the behaviour of the short end of the curve suggested by the Taylor rule (see, Taylor (1993)), and the fact that the long end of the curve is determined by market participant's expectations to long-term inflation, economic growth and a risk premium.

- Regime switches help identify generic yield curve shapes, that are necessary for making long term yield curve projections.

- A modelling framework as suggested here might be thought of as a short hand way of obtaining a practical link between macro economic scenarios and the yield curve shape and location without the burden of finding the optimal set of variables to condition the yield curve estimation on, i.e. it represents a reduced form model.

The observation equation

The vector of yields Y at time t for different maturities $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ can be expressed as a function of yield curve factors and yield curve factor sensitivities.

$$Y_t(\tau) = H\beta_t + \varepsilon_t, \tag{7.10}$$

where H collects the Nelson-Siegel factor sensitivities:

$$H = \begin{bmatrix} 1 & \frac{1-e^{-\lambda\tau_1}}{\lambda\tau_1} & \frac{1-e^{-\lambda\tau_1}}{\lambda\tau_1} - e^{-\lambda\tau_1} \\ 1 & \frac{1-e^{-\lambda\tau_2}}{\lambda\tau_2} & \frac{1-e^{-\lambda\tau_2}}{\lambda\tau_2} - e^{-\lambda\tau_2} \\ \vdots & \vdots & \vdots \\ 1 & \frac{1-e^{-\lambda\tau_n}}{\lambda\tau_n} & \frac{1-e^{-\lambda\tau_n}}{\lambda\tau_n} - e^{-\lambda\tau_n} \end{bmatrix}, \tag{7.11}$$

and β collects the yield curve factors i.e. the parameters to be estimated:

$$\beta_t = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}_t = \begin{bmatrix} \text{yield curve level} \\ \text{yield curve slope} \\ \text{yield curve curvature} \end{bmatrix}_t. \tag{7.12}$$

It is assumed that the residual volatility is $\varepsilon_t \sim N(0, R)$.

The state equation

The state equation describes how the unobserved factors evolve over time. Here it is assumed that a VAR(1) with regime switching in the means is appropriate. Hence:

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}_t = \begin{bmatrix} m_1^I \\ m_2^J \\ m_3^K \end{bmatrix} + \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}_{t-1} + v_t, \tag{7.13}$$

where $v \sim N(0, Q)$. The superscript on the mean parameters indicate regime affiliation. In our example of the model, however, only the slope is allowed to exhibit regime switching behaviour i.e. $I = \{I\}$, $J = \{a, b, c\}$, $K = \{K\}$. The reason for this is substantiated by theory and empirical studies suggesting that of the three factors the slope is most strongly related to economic activity. Also a pragmatic approach suggests that regime switches are best suited for the slope parameter: using any kind of recent yield curve data from US or Europe shows that the long term level has decreased steadily over time. In fact visual inspection of a time series of 10 year yields will not reveal any apparent structural breaks but rather an evolvement close to a random walk with a negative trend. The curvature explains relatively little of the variation of yields over time (approximately around 5%), so even though there might be regime switches in the curvature factor it is questionable how much value they would add if modelled.

Estimation procedure

This section shows how the model presented above can be estimated. As stated, it is assumed that only one underlying factor exhibits regime-switching behaviour and that the mean captures this switching behaviour. The likelihood estimation of the model is based on a combination of the kalman filter and Hamilton regime-switching algorithms, as illustrated by Kim and Nelson (2000); for a detailed presentation of the Kalman filter see e.g. Hamilton (1994, ch.13) and for a good presentation of regime-switching models see e.g. Hamilton (1994, ch.22).

With regime-switches the state equation takes the following form

$$\beta^j_{t|t-1} = m^j + F\beta_{t-1|t-1} + e_t, \tag{7.14}$$

where $j = (1, 2, \ldots, S)$ indicates regimes, m is the mean, F is the matrix of autoregressive parameters, and $\beta_{t-1|t-1}$ is a probability weighted average of the betas from the previous kalman filter iteration. The observations equation translates the unobserved factors from (7.14) into yields:

$$Y_t = H\beta^j_t + v_t, \tag{7.15}$$

Y_t is the vector of yields observed at time t for the desired n maturities and H is a matrix comprising the Nelson-Siegel factors, as defined in (7.11).

The prediction errors η of the kalman filter will be regime-dependent as a consequence of the regime dependence of $\beta_{t|t-1}$ as illustrated in (7.14),

$$\eta^j_{t|t-1} = Y_t - H\beta^j_{t|t-1}. \tag{7.16}$$

It is assumed that the possible j regimes affect only the mean-specification of β, so variance terms are implicitly assumed to be unaffected by changes in regime. For this reason the conditional prediction variance in the kalman filter is expressed as,

$$f_{t|t-1} = E\left[\eta^2_{t|t-1}\right] = HP_{t|t-1}H' + R \tag{7.17}$$

where $P_{t|t-1}$ is the one-step ahead predictor for the conditional covariance of β_t, which is

$$P_{t|t-1} = FP_{t-1|t-1}F' + Q. \tag{7.18}$$

The density at time t for each of the j regimes can then be calculated by

$$l^j_t(\theta) \propto \left|f_{t|t-1}\right|^{-0.5} \exp\left\{-\frac{1}{2}\eta^j_{t|t-1}f^{-1}_{t|t-1}\eta^j_{t|t-1}\right\}. \tag{7.19}$$

To complete the kalman filter, β as well as its variance need to be updated. While the updating of the variance proceeds according to the regular Kalman Filter since the regime classification does not appear in (7.18), updating β has to take into account the regime-switching as given in (7.14). As in Kim&Nelson the updated β is calculated as a weighted average of the predicted β, using as weights the state probabilities from the regime switching filter. These probabilities π are calculated as:

$$\pi_{t|t} = \frac{\pi_{t|t-1} \odot D_t}{1'(\pi_{t|t-1} \odot D_t)}, \tag{7.20}$$

where $\odot$ is the element-by-element multiplicator, D collects the densities in a row vector, and 1 is a vector of ones of dimension j. According to the Hamilton filter these probabilities are predicted one-step ahead in the following way:

$$\pi_{t+1|t} = p\pi_{t|t-1}, \tag{7.21}$$

with p being the transition probability matrix. The Kalman Filter iteration can then be completed by updating P in the regular way and β through a weighted average of the updated values.

$$P_{t|t} = P_{t|t-1} - P_{t|t-1}H' f_{t|t-1} H P_{t|t-1} \tag{7.22}$$

$$\beta^j_{t|t} = \beta^j_{t|t-1} + P_{t|t-1}H' f^{-1}_{t|t-1} \eta^j_{t|t-1} \tag{7.23}$$

$$\beta_{t|t} = \pi_{t|t} B, \tag{7.24}$$

where B collects $\beta^j_{t|t}$ for $j = (1, 2, \ldots, S)$ and $\beta_{t|t}$ thus can be seen as a weighted average of the individual regime betas.

The log likelihood function to be maximised is actually the log of the denominator of (7.20), i.e. the log of the weighted sum of the densities for each of the regimes:

$$LogL(\theta) = \sum_{t=1}^{T} \log\left[\mathbf{1}'(\pi_{t|t-1} \odot D_t)\right],$$

where $\mathbf{1}$ is a vector of ones (just used to sum the state densities), D_t collects the densities for each state j at time t, $\pi_{t|t-1}$ is a j by 1 vector holding the estimated state probabilities at time t given information as of time $t-1$, and $\odot$ is the element-by-element multiplicator.

As illustrated above the maximum likelihood estimation problem is solved by combining the kalman filter and the Hamilton filter. By iterating through data the likelihood of each factor being drawn from the j different possibilities defined by the state is evaluated. This is accomplished by expanding the "dimension" of β in the kalman filter to fit the number of states modelled, and at the end of each iteration by collapsing the "dimension" to one through a weighted average, where the weights are the regime probabilities that are derived by the Hamilton filter.

Estimation example

The yield curve modelling framework described above is implemented in Matlab via the following functions. The data needed to estimate the model is collected in an Excel(tm) sheet called *yld2beta.xls*. Since there is a fair amount

of data needed to run the tool, it is more convenient to collect the data in Excel and then send it to Matlab using the Matlab(R) Excel add-in. The three necessary Matlab(R) files are contained on the cd accompanying the book and they are called *yld2beta_shell.m*, *yld2beta.m*, and *yld2beta_likeli.m*. The first of these files is organising data into a structured variable after which it calls the main calculation program (*yld2beta.m*). The likelihood function is programmed in the file labelled *yld2beta_likeli.m*, which is invoked from *yld2beta.m*.

Since the Matlab(R) implementation of the model follows very closely the formulas given above it seems more fruitful to leave it to the reader to go through the files. Instead, we will show some of the results that are generated when running the *yld2beta_shell* program.

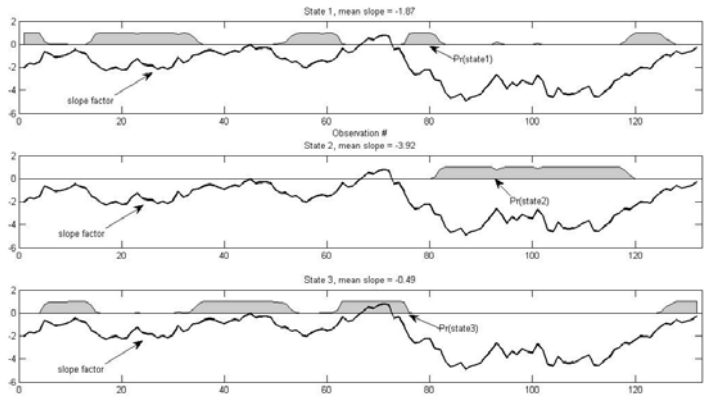


Figure 7.3: State probabilities and the slope factor

As can be seen in Figure 7.3 three distinct regimes are identified. The probability for each state is shown as a shaded area in the figure and the slope factor is plottet to serve as a point of comparison. Regime 1 is identified by slope observations having a positive slope of 1.87, i.e. 187 basis points. It is recalled that the slope factor in the Nelson-Siegel model is defined as minus the "traditionally" used slope convention, so a negative slope in the Nelson-Siegel yield curve models corresponds to an upward sloping yield curve. The second regime comprises steeply upward sloping yield curves as identified by a mean slope factor of 392 basis points. The last regime 3 comprises yield curves that are close to being flat, and some are even inverse, generated by

a mean slope of roughly 49 basis points. It is further noted from the Figure that the regimes are clearly identified by probabilities close to unity when the regime is active and zero when the regime is inactive.

7.5 Importance Sampling

Books and journal articles have been written about how to perform Monte Carlo studies in the most clever ways possible.[6] This flourishing literature should probably be seen against the prolifiration of complex options and the need to quickly and precisely price such instruments. A key concept in the design of clever Monte Carlo studies is how to reduce the variance of the estimator and there are different ways of doing this, e.g. via antithetic sampling, control variates, stratified sampling and so forth.[7] It is well beyond the scope of this chapter to treat all these different techniques and therefore we choose to focus on one technique that is of particular relevance to the estimation of risk measures, namely importance sampling. As it is demonstrated below, in an asset allocation context, importance sampling is particularly relevant for the calculation of tail risk measures and statistics based on rare events.

7.5.1 Some theory

The basic idea of importance sampling is to focus ones attention on the parts of the distribution of the variable, that is of greater interest for the task at hand. For example, if we want to estimate the expected shortfall of a bond portfolio by the use of simulation techniques, a more precise estimate will be obtained if we can generate more realisations in the loss tail of the distribution than would occur under normal circumstances e.g. 1% of the time if we look at the expected shortfall at a 99% level of significance. So, we need a technique that generates and over-representation of tail outcomes and at the same time tells us how to adjust the resulting simulation based estimate for this generated degree of tail over-representation. One such answer is provided by the importance sampling technique; Glasserman (2004, p. 255 and Appendix B.4) reviews the theoretical underpinnings of the methodology.

Let $h(x)$ represent a functional transformation of the variable x that is needed for calculating the metric that we are interested in. For example, if in-

[6] For good books in this arena see for example Glasserman (2004) and Jackel (2002).

[7] In this context the term "clever" is used as an antonym to "brute force" calculations.

terest is on estimating the expected shortfall then $h(x) = x * I_{x \leq VaR_{1-a}} * \frac{1}{1-a}$, with the expected shortfall being calculated by: $\widehat{\theta}_{ES} = \frac{1}{1-a}\frac{1}{T}\sum_{t=1}^{T} h(x_t)$. Here x represents the simulated variable, T is the sample size, a is the confidence level, and VaR_{1-a} is the Value-at-Risk level for x, and I is an indicator function taking on the value 1 if x_t is in the tail and 0 otherwise. To simulate the T realisations of the variable x, an application of the traditional (brute force) Monte Carlo technique stipulates that random draws are generated from a probability density function f. If innovations, e_t, to the process governing x are normally distributed with variance σ_e^2, we would draw random numbers from $N(0, \sigma^2)$ and feed them trough the appropriate calculation expression to generate realisations for x. In this case $f(x)$ is the probability density function of the normal distribution.

Rather than using the density f to generate realisations for x, the importance sampling technique suggests to use a density g. The g density should be different from f in a way to ensures that more realisations of x take on interesting values e.g. that more realisations of x fall in the loss-tail, if we want to estimate expected shortfall.[8] Furthermore, the ratio of the two densities f and g is used to adjust the calculated summary statistic for the over-reprasentation of "interesting" observations generated by g. In effect, the importance sampling estimate is calculated by:

$$\widehat{\theta} = \frac{1}{T}\sum_{j=1}^{T} h(x_j)\frac{f(x_j)}{g(x_j)}, \tag{7.25}$$

Sometimes g is referred to as the "shifted" density and the ration f/g is called the density ratio (or the Radon-Nikodym derivative or the measure change). In equation (7.25) the term $h(x_j)$ accounts for the function evaluation of the *shifted* x variable i.e. the values for x based on the simulation using the shifted g density. The numerator $f(x_j)$ represents the evaluation of x under the original density, and $g(x_j)$ represents the evaluation of x under the shifted density.

7.5.2 An example

To illustrate how importance sampling can be implemented consider the Matlab(R) script below. It calculates the expected shortfall for a bond portfolio

[8] The choice of g has to satisfy that $f(x) > 0 \Rightarrow g(x) > 0$, see Glasserman (2004, p. 255)

and illustrates that using the importance sampling technique can reduce the variability of the parameter estimate.

```
[1] nObs = 250;
[2] nIter = 500;
[3] a = 0.99;
[4] f = zeros(nObs,1);
[5] g = zeros(nObs,1); %
[6] C = [ 18.6 15.6 17.9 15.8 11.3;
[7]       15.6 21.4 19.6 19.9 10.9;
[8]      17.9 19.6 36.3 28.2 15.8;
[9]      15.8 19.9 28.2 35.8 16.1;
[10      11.3 10.9 15.8 16.1 38.6 ];
[11] w = [ 0.51; 0.29; 0.00; 0.02; 0.18 ];
[12] iShift = -10;
[13] %
[14] % Calc theoretical VaR
[15] %
[16] VaR_base = -norminv(a)*sqrt(w'*C*w);
[17] ES_base = exp(-(norminv(a)^2)/2)/((1-a)*(2*pi)^0.5)*sqrt(w'*C*w);
[18] ES_S = zeros(nIter,2);
[19] %
[20] % simulating portfolio returns
[21] %
[22] for ( j=1:nIter )
[23]     disp( sprintf('Percentage completed %3f', 100*j/nIter) )
[24]     eps1 = randn(nObs,length(C))*chol(C);
[25]     rp = eps1*w;
[26]     rp_IS = (eps1+iShift)*w;
[27]     for ( k=1:nObs )
[28]         f(k,1) = exp(-0.5* ( rp_IS(k,1)^2/(w'*C*w) ) );
[29]         g(k,1) = exp(-0.5* (( (rp_IS(k,1)+iShift)-mean(rp_IS(:,1)+iShift) )^2/(w'*C*w)
));
[30]     end
[31]     ES_S(j,1) = -mean([rp<=VaR_base].*rp) * 1/(1-a);
[32]     ES_S(j,2) = -mean([rp_IS<=VaR_base].*rp_IS .* (f./g) ) * 1/(1-a);
[33] end
[34] disp('Means')
```

```
[35] mean(ES_S)
[36] disp('Variances')
[37] var(ES_S)
[38] disp('Theoretically calculated expected shortfall')
[39] ES_base
[40] figure
[41] subplot(1,2,1), plot(ES_S(:,1),'k-'), title('Standard Estimate'), ...
[42]                                 axis([0 nIter 0 max(ES_S(:,1))])
[43] subplot(1,2,2), plot(ES_S(:,2),'k-'), title('Importance Samp.  Est.'), ...
[44]                                 axis([0 nIter 0 max(ES_S(:,1))])
```

Running the above script gives the following results: Figure 7.4 draws a comparison between the estimate of expected shortfall using "brute force" (the left graph) and using importance sampling (the right graph) for each of the 500 iterations specified in line [2]. The y-axis scales in the two plots are identical and the variance reduction obtained by using importance sampling is thus striking.

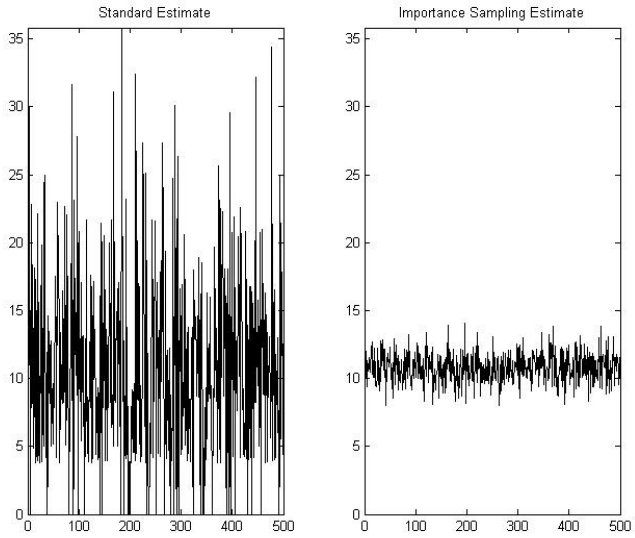


Figure 7.4: Estimates of Expected Shortfall

The estimated mean of the expected shortfall estimations are both identical

to the therotical value of 10.8, however the variance of the "brute force" calculation is 46.6 compared to a variance of 1.2 for the importance sampling technique. Hence, the latter can facilitate a variance reduction of approximately a factor 50 using the example data above. And, this reduction is obtained without optimising the shifted distribution. In the end-chapter exercises the reader is invited to find the optimal shift factor for the concrete example presented in the Matlab(R) script file.

Turning now to the example script file. Lines [1]-[12] allocate starting values to variables. Line [1] specifies the number of observations to be generated for each simulated portfolio return distribution, and line [2] specifies the number of times a portfolio return distribution is simulated. In Figure 7.4 the x-axis is defined from 1 to $nIter = 500$, i.e. covering the number of times the estimation experiment is repeated, and each observation in the graphs (one in each panel) is the resulting expected shortfall calculated on the basis of $nObs = 250$ simulated portfolio returns. Line [3] sets the confidence interval, which is this case is the 99% level. Line [4] and [5] prepare vectors to contain the density values, f for the unshifted and g for the shifted density. Lines [6]-[10] defines the covariance matrix, line [11] the portfolio weights, and line [12] the size of the mean shift for density g. Line [12] sets $iShift = -10$, which is close to the 99% Value-at-Risk using the covariance matrix C. This seems to be a qualified guess for the size of the shift, since the VaR marks the begining of the tail relevant for the expected shortfall calculations. Lines [13]-[18] calculate the VaR and the theoretical expected shortfall value using (4.1) and (4.2). Effectively, VaR_base contains the 99% Value-at-Risk and ES_base contains the 99% expected shortfall assuming that the underlying data is normally distributed. Line [18] prepares the vector, ES_S, as a container for the calculated expected shortfalls. Lines [19]-[34] are responsible for the main calculations performed by the script. An outer loop over the variable $nIter$ is spanned by lines [22]-[33]. The calculation inside this loop thus constitutes one single calculation of the expected shortfall on the basis of a "brute force" Monte Carlo method and the importance sampling technique. Using simulated data the results are stores in the vector called ES_S; the calculation and storing takes place in lines [31] and [32]. Line [31] is the "brute force" calculation applying the function $average\left(r_p * I_{r_p \leq VaR_{1-a}}\right) * \frac{1}{1-a}$. Line [32] performs a similar calculation using however the shifted density. Therefore the estimate is corrected by the density ration f/g as described above. Lines [28]-[31] calculate the unshifted and shifted densities on the basis of the normal distribution. As an information to the user, line [23] prints to

screen the percantage completed simulations. Line [24] generates correlated random numbers using the Cholesky decomposition, as we have seen before, and lines [25]-[26] calculate the unshifted and shifted portfolio returns, respectively. The remaining lines [34]-[42] reports the results.

Chapter 8

Building graphical user interfaces

8.1 Introduction

This chapter gives a general introduction on how to construct Graphical User Interfaces (GUIs) in Matlat. Many possibilities and features are available to this end, and the current chapter only scratches the surface. Following the general spirit and intention of this book, the current chapter will illustrate the GUI capabilities of Matlab via an example. In particular, we will build a front-end that allows the user to explore the Nelson and Siegel (1987) model as reparameterized by Diebold and Li (2006).

Learning objectives

- To be able to use the Matlab(R) development environment for Graphical User Interfaces (GUI)
- Learn the central elements of a Matlab(R) GUI building feature (guide)
- How to generate input and output from a GUI
- How to pass variables between GUI functions

8.2 The "guide" development environment

The most straight-forward way to build a GUI, and the method we will use, is to initiate the Matlab(R) "guide". This is a user-friendly graphical environment that will make the most vital elements of GUI building available to us at the click of a button. To start the guide, simple type

```
[1] guide
```

at the command prompt. This will bring up the following screen:

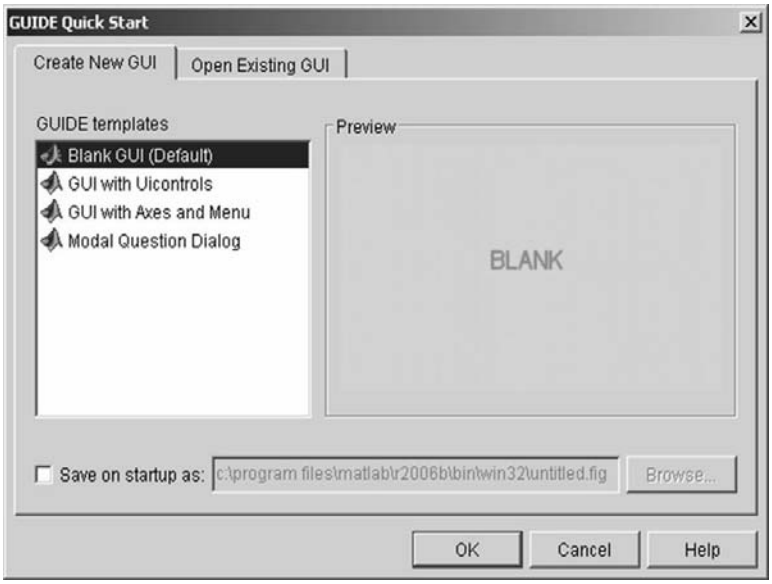


Figure 8.1: The Matlab GUIDE start screen

Clicking "OK" to the choice activated by default in the start screen will open the development environment in which we will develop out "Nelson-Siegel Example" GUI.
To the left in the GUI development area (see 8.2) are the varios controls / callbacks that we can add to the GUI. For the example GUI that we are going to build the following are needed:

- a "Push Button" allows us to execute a command when the button is activated;

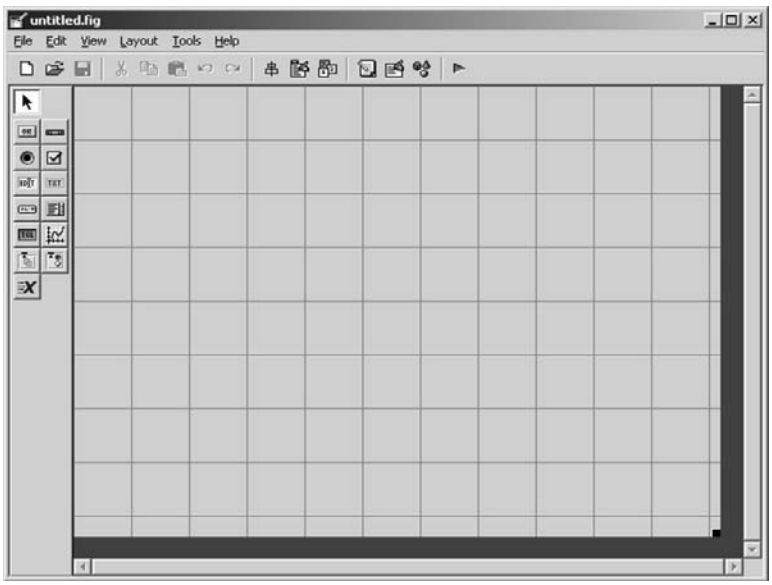


Figure 8.2: The GUI development area

- an "Edit Text" box can be used to take a single input from the user;
- the "Static Text" box gives us the possibility to write headings and messages to the user;
- the "Axes" box lets us print a figure in the GUI. Alternatively, a figure can be created outside the GUI;
- the "uitable" feature is not shown on the GUI development area. It allows for easy input and output to the GUI.

An additional comment is warranted on the last bullet-point mentioned above. The "uitable" is an extremely useful feature for data transport to and from the GUI. Based on rows and columns dimensions specified by the user, it installs an Excel(tm) like grid on the GUI. This is far to be preferred over for example an "Excel sheet" ActiveX component, because such a component will only work if a compatible version of Excel is installed on the machine where the GUI is executed. Therefore, the "uitable" is more general and will give rise to less problems if the GUI, as it is often the case, is intented for use on many different PCs. Unfortunately, "uitable" seems not

to be supported by Matlab(R): there is no help function directly available[1]. Nontheless, we will still make use of the "uitable" in this chapter.

Another general issue to mention is how to use the "Property Inspector", which is activated by double-clicking on the "controls", and which allows the GUI designer to change general features of that particular contral. We can illustrate the "Property Inspector" by way of example by changing the name of our GUI. Currently in is called "untitled.fig" (see the upper left corner of 8.2). Lets change this to something that says more about the purpose of the GUI, for example "Nelson-Siegel Example". To do this we go through the following steps:

- double-click on the GUI area i.e. the grey area in 8.2;
- this brings up the following pop-up window;
- scroll down to the "Name" property and change it to "Nelson-Siegel Example";
- close the "Property Inspector" as normally in windows(tm) by clicking the "x" in the upper left hand hand corner;
- save the GUI by clicking on the "save button" on the GUI design area. This will save figure file and bring up a Matlab(R) m-file that holds the code of the GUI. This file, called "NelsonSiegelExample.m", contains some autogenerated code that we will, and should in general, not touch. However, below the auto-genearted code, we can add the code that eventually will form the functionalities of our GUI.

8.3 Creating a simple GUI

We are now ready to populate the GUI with the controls that we need. For each control we add to the GUI, Matlab(R) automatically generates the appropriate "header code" to the "NelsonSiegelExample.m" file, i.e. Matlab(R)

[1] Writing "help uitable" in the Matlab(R) command prompt gives the following message: "*WARNING: This feature is not supported in MATLAB and the API and functionality may change in a future release*". The help text for the "uitable" can be found by writing "edit uitable", i.e. by opening the source code for the "uitable" in the Matlab(R) editor. Beware of not changing the code when doing this!

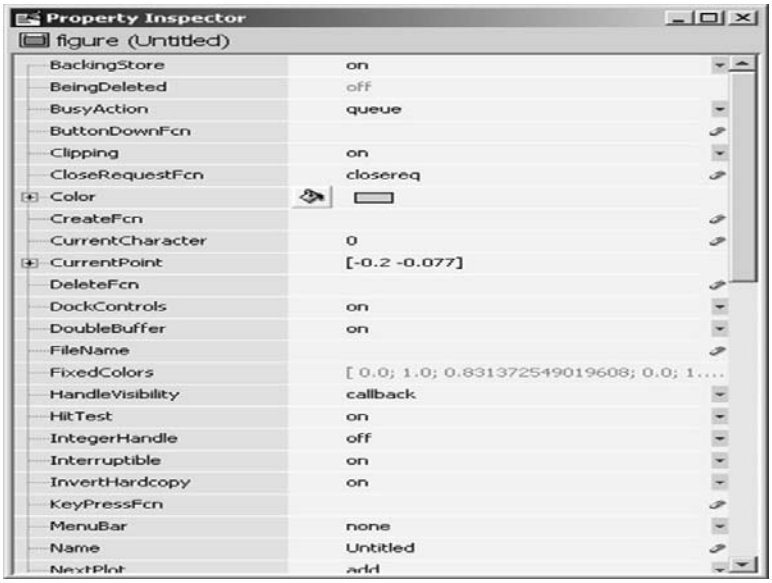


Figure 8.3:

will add the necessary function statements that we can then afterwards modify to give e.g. a "Push Button" its desired features. For the purpose of the GUI we need the following controls:

- three "Push Buttons";
- two "Axes" plotting areas;
- seven "Edit Text" boxes" and nine "Static Text" boxes;
- two "uitables" (not seen in the GUI menu);
- two "Button Groups" and four "Radio Buttons";
- and, three "panels" to keep the functionalities of the GUI separate.

All these controls can be added to the GUI design area by dragging-and-dropping them from the left hand side menu, apart from the "uitable", which we will add manually at a later stage. In a first step, lets add the necessary controls to the GUI design area, re-name and re-size them. For the "Push Button" the text displayed on the button it-self can be changed

in the "Property Inspector" under "String". It is also be a good idea to change the "Tag" of the "Push Button". The tag is the name that the button is given in the function that Matlab(R) automatically adds to the "NelsonSiegelExample.m" when a button is added to the GUI. Changing the tag makes it easier to identify the push buttons and the other controls once we start adding functionalities to GUI. The name and tag change is shown in 8.4 for the first push button we add to the GUI.

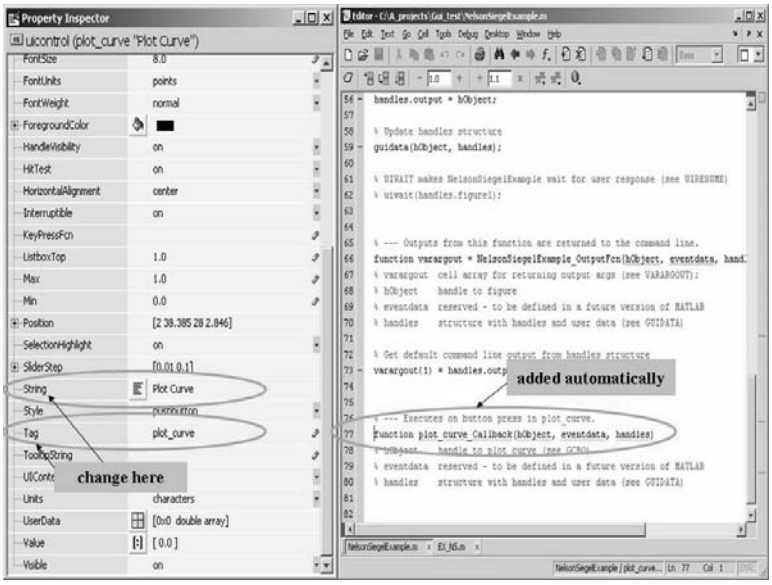


Figure 8.4: Changing the Name and Tag properties of a "Push Button"

Once all controls are added to the GUI it should look like 8.5.

The next step is to add functionalities to the GUI. This is done by adding code to the "NelsonSiegelExample.m". By now, Matlab(R) has already added functions calls for each of the controls we have added to the GUI to this file. Our job is to add the specific lines of code to this pre-generated structure as to ensure that the GUI indeed does the job we want it to do. One important feature we need in this connection is to be able to pass data and variable values between the functions calls of the GUI. We can do this by using the following commands in the GUI code:

```
[j] handles.var1 = 1;
```

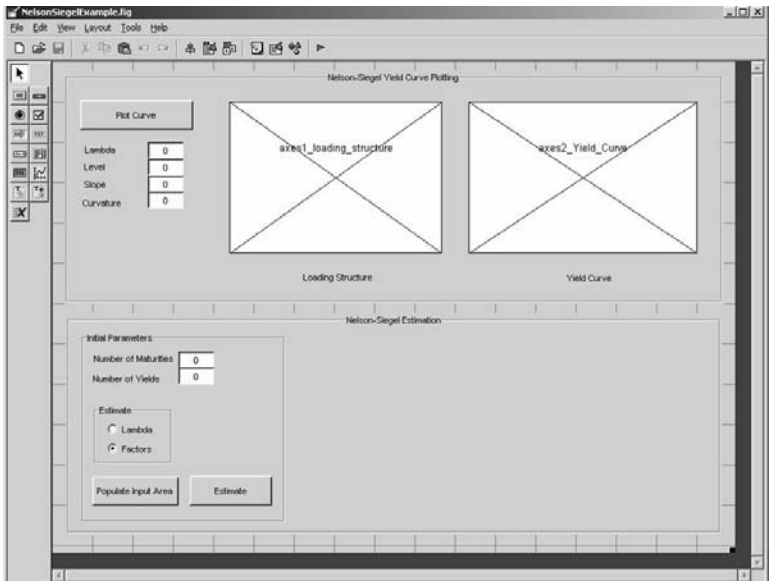


Figure 8.5: Nelson-Siegel GUI

```
[k] guidata(hObject,handles);
[y] handles = guidata(hObject);
[z] temp    = handles.var1;
```

. . . . . .

Line [j] creates a new variable in the structured variable called "handles". The new variable called "var1" is given the value 1. Line [k] stores the variable "handles" in the guidata register under the object called "hObject". The variables storred in "handles" can then be recalled and operations can be performed on them in other functions in the GUI by using line [y] to recall the "handles" structure and by allocating the values storred in the handles structure, as e.g. shown in line [z]. This process will be illustrated in detail when generating the code for our example GUI.

8.3.1 Plotting the yield curve

Our Nelson-Siegel GUI is devided into two main areas. In this section we will describe how to get the first of these, labelled "Nelson-Siegel Yield Curve Plotting" up-and-running. The purpose of this part of the GUI is to allow the user to plot Nelson-Siegel yield curves. For illustrative purposes the GUI

also plots the factor loading matrix implied by the Nelson-Siegel model, in particular by the choice of the λ parameter. Based on the inputs: level, slope, curvature and λ, the GUI plots the the factor loading structure in the first graphing area and the Nelson-Siegel yield curve in the second graphing area and. In other words, following equation (7.10), the first plotting area displays $Y_t(\tau)$ on the basis of the input supplied by β_t and λ, from equation (7.11); the second plotting area displays H from (7.11).

First we need to store the input values entered in the four edit boxes for Lambda, Level, Slope, and curvature. The autogenerated code by Matlab(R) for the edit box callback consists of two parts as illustrated below.

```
[1] function edit1_Callback(hObject, eventdata, handles)
[2] % hObject handle to edit1 (see GCBO)
[3] % eventdata reserved - to be defined in a future version of MATLAB
[4] % handles structure with handles and user data (see GUIDATA)
[5]
[6] % Hints:  get(hObject,'String') returns contents of edit1 as text
[7] % str2double(get(hObject,'String')) returns contents of edit1 as a double
[8]
[9] % --- Executes during object creation, after setting all properties.
[10] function edit1_CreateFcn(hObject, eventdata, handles)
[11] % hObject handle to edit1 (see GCBO)
[12] % eventdata reserved - to be defined in a future version of MATLAB
[13] % handles empty - handles not created until after all CreateFcns called
[14]
[15] % Hint:  edit controls usually have a white background on Windows.
[16] % See ISPC and COMPUTER.
[17] if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
[18] set(hObject,'BackgroundColor','white');
[19] end
```

Line [1] contains the auto-generated function call for the first edit-box. Lines [2]-[7] contain information text that is not executed when the code is run because all lines are preceded by a %-sign that by Matlab(R) is interpreted as a text comment. Lines [8]-[12] are left blank to indicate that these lines (and more if needed) are used to enter the functionalities of the edit-box. Lines [13]-[23] also contain auto-generated code relating to the edit-box. We will not comment on or edit in this part of the code.

The purpose of the input box called "edit1" is to take user input for the value of λ and to store it in the guidata set. How to do this is shown below.

```
[1] function edit1_Callback(hObject, eventdata, handles)
[2] handles.Lambda = str2double(get(hObject,'String'));
[3] guidata(hObject,handles)
```

Again, line [1] is the function call. As seen above, line [2] generates a new variable, in this case called "Lambda", and adds this variable to the structure "handles". The value keyed into to the editbox is transferred to the variable through the `get(hObject,'String')` command. The keyword "string" indicates that the datatype of the input box is text. To convert the text into a number we use the `str2double` command. Similar lines of code are to be repeated for each of the input box call-back functions, whereby the variables: "Level", "Slope", and "Curvature" are generated and added to the "handles" data structure.

After the input variables have been collected and their values added to the data structure we can complete the first part of the GUI that prints Nelson-Siegel curves. The code below shows how.

```
[1] function plot_curve_Callback(hObject, eventdata, handles)
    [2] % ...  Organising data
    [3] handles = guidata(hObject);
    [4] Lambda = handles.Lambda;
    [5] Level = handles.Level;
    [6] Slope = handles.Slope;
    [7] Curvature = handles.Curvature;
    [8] % ...  Constructing NS curve and sensitivities
    [9] nTau = 120;
    [10] tau = [1:1:nTau]';
    [11] H = [ ones(nTau,1) (1-exp(-Lambda.*tau))./(Lambda.*tau) ...
    [12]     (1-exp(-Lambda.*tau))./(Lambda.*tau)-exp(-Lambda.*tau) ];
    [13] Y = H*[Level;Slope;Curvature];
    [14] % ...  Plotting results
    [15] axes(handles.axes1_loading_structure); % Plot for H matrix
    [16] plot(H);
    [17] axis([1 120 0 1.2]);
    [18] set(gca,'XTick',[12:24:120]);
```

```
[19] axes(handles.axes2_Yield_Curve); % Plot for Y
[20] plot(Y,'LineWidth',2);
[21] axis([1 120 0 Level+1]);
[22] set(gca,'XTick',[12:24:120]);
```

Line [1] defines the function name according to our choice made for the "tag" in the property inspector related to the "push button" call-back on the GUI form. Line [1] is generate automatically by Matlab and should in general not be changed. What we need to do, though, is first is to organise the data useful for the drawing of the Nelson-Siegel yield curve. The relevant variables are stored in the "handles" structure and lines [3]-[7] extracts user-input values. Line [3] transfers the handle structure to the function so it can be used further. Lines [4]-[7] subsequently allocates the user-input values to variables given meaningful names, namely " Lambda", "Level", "Slope", and "Curvature". It can be observed how the "handle" structure has allocated a field to the value of each variable and how the these values can be recovered and reassigned to another variable, as also illustrated above. Line [9] specifices a variable called "nTau" which will hold the number of maturity observations that we want to include in the plots. In principle, "nTau" could also be an input variable, but since its value will only affect the smoothness of the plotted curves, and since a value of 120 will be adequate for any practical purposes there seem to be little rational for asking the user to specify its value. "nTau" is therefore a constant decided by the programmer. Line [10] generates an nTau-by-1 vector of observations covering the values 1 to nTau with a step length of 1. Line [11] contains the formula for the Nelson-Siegel sensitivity matrix called "H". The used formula is recognised from equation (5.2). Following equation (5.1), the expected Nelson-Siegel curve can be calculated when assuming that vector $\varepsilon_t = 0$. This is done in line [13] and the yields for the 120 maturities are stored in the vector called "Y". The only remaining issue to be dealt with is to plot the H matrix and the Y vector in the designated plotting areas on the GUI. Such areas are called "axes" and lines [15]-[16] show how to plot the factor sensitivity matrix to the first axes area we generated on the GUI. Usually, in Matlab we start by using the "figure" command before the "plot" function to ensure that the plot we create does not overwrite a previous plot. Similarly, in the GUI we first activate the axes that we want to plot to, and then we perform the plot. Line [15] "activates" the plotting area we generated to hold the factor sensitivity plot. This area was given the name "`axes1_loading_structure`" which

is contained by the "handles" structure. Line [16] then plots H in the "activated" axes. Lines [17] specifies the x- and y-axes, defining the x-axis to span the values from 1 to 120 (for the maturities) and the y-axes to span 0 to 1.2. Line [18] concludes the plot by setting the tick size of the x-axes to values from 12 to 120, with a step size of 12. Since the maturity is denominated in months it improves the readability of the plot of this is done. The remaining lines of the above function: lines [19]-[22] completes the plot by allocating to the second axes ares the calculated Nelson-Siegel yield curve values. This completed the first part of our Nelson-Siegel example GUI. An example of how this part of the GUI can be used in given in Figure 8.6.

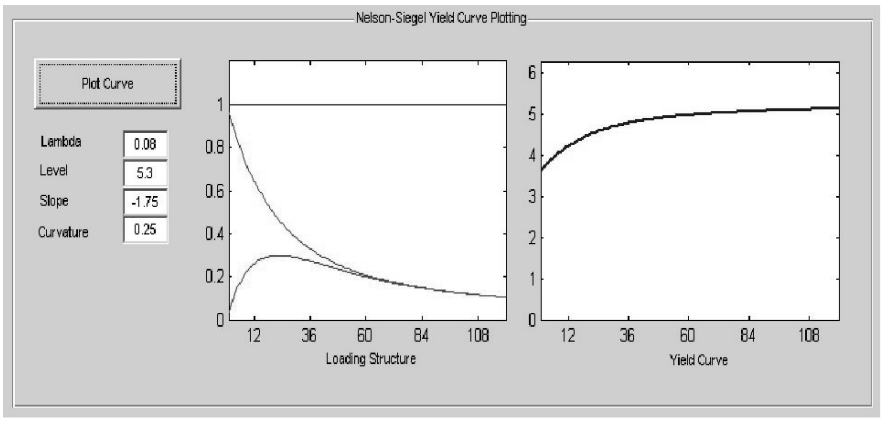


Figure 8.6: Illustrates the usage of the first part of the Nelson-Siegel example GUI

8.3.2 Estimating λ and yield curve factors

To complete our GUI we need to generate a function that allows for estimation of the Lambda variable as well as the Nelson-Siegel yield curve factors. Based on the User's choice to estimate either "Lambda" or "Factors" the user needs to supply yield curve data as well as the for which maturities the yield curve data are observed. To this end, the lower part of the Nelson-Siegel example GUI takes four initial inputs. This input area is highlighted below. The inputs "Number of Maturities" and "Number of Yields" helps to define the input area further. In order for the tool to be able to either estimate the "Lambda" coefficient or the Nelson-Siegel factors, it needs yield curve

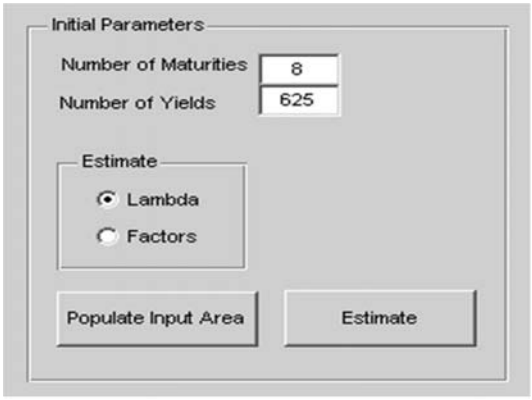


Figure 8.7: Input area for the lower part of the Nelson-Siegel example GUI.

data. The inputs given for these two variables basically define the size of the data matrix to contain the yield curve observations and the length of the vector to hold the maturities at which yields are observed. To store the values entered in the two input boxes is equal to that explained above and it is hence omitted here. The second part of the input area defined on the basis of the inputs in "Number of Maturities" and "Number of Yields". In particular, the "push button" call-back "Populate Input Area" activates two uitables of appropriate dimensions. The relevant code is shown below.

```
.....................................................................
[1] function Populate_Input_Area_Callback(hObject, eventdata, handles)
[2] handles     = guidata(hObject);
[3] nMaturities = handles.nMaturities;
[4] nYields     = handles.nYields;
[5] dat_mat     = nan(nMaturities,1);
[6] dat_Y       = nan(nYields,nMaturities);
[7] dat_lambda_full = nan(1,3);
[8] for ( j=1:nMaturities )
[9]     colnames(1,j) = cellstr(strcat('Y_tau ', num2str(j)));
[10] end
[11] % ...  populate input area
[12] uicontrol('Style','text','String','Enter Lambda', 'Position', [315 260 80 15]);
[13] uitable_lambdas           = uitable(dat_lambda_full, {'Lambda', 'Lower', 'Upper'},
'Position', [310 220 250 35]);
```

```
[14] uicontrol('Style','text','String','Enter Maturities', 'Position', [315 195 80 15]);
[15] uitable_maturities        = uitable(dat_mat, {'tau'}, 'Position', [310 40 105 150]);
[16] uicontrol('Style','text','String','Enter Yields', 'Position', [600 195 100 15]);
[17] uitable_Yields            = uitable(dat_Y, colnames,'Position', [430 40 385 150]);
[18] handles.uitable_lambdas   = uitable_lambdas;
[19] handles.uitable_maturities= uitable_maturities;
[20] handles.uitable_Yields    = uitable_Yields;
[21] guidata(hObject,handles)
```

Line [1] defines the function name, line [2] loads the "handles" data into the function, and lines [3]-[4] are standard by now. Lines [5] generates a vector that will hold the user defined maturities at which yields are observed and line [6] similarly defines a matrix to hold the actual yield curve observations. It is see that both these input variables are initially given the "nan" values. And, line [7] genrates a vector to hold the starting and upper and lower bounds for the "Lamda" coeficient, if the user chooses to estimate "Lambda". Lines [8]-[10] generates header information for the maturity observations. Lines [12]-[17] generate the uitables necessary for storing the additional inputs comprised by maturities, yield curve observations, and the starting value for lamba together with upper and lower bounds on "Lambda". Line [18]-[20] stores the input in the "handles" structure, and line [21] finally stores the handle structure for later use. For each uitable input we also generate a title. One needs to play around with the positioning of the title and the uitable to get it right. The general principle we use is to define the title via a "text" "uicontrol" as illustrated in e.g. line [14]. Then the position and size of the uitable is defined accordingly, as e.g. in line [15]. This particular area is generated to hold the vector of maturities. The numbers that are entered into this uitable are stored in the variable called `uitable_maturities` by setting this variable equal to the uitable command. The first input to the uitable is the prespecified data vector, which we called dat_mat, the next input is the title, in this case we use the title "tau"; next a keyword is given "position" which indicates that what follows defines the position of the uitable. The position is defined by four numbers: (1) distance to left boundary of the GUI, (2) distance to the floor of the GUI, (3) then the hight, and finally (4) the width.

If we press the button "Populate Input Area" the above code is activated and generates the following expanded input area.
By now we know the major building blocks necessary to construct user

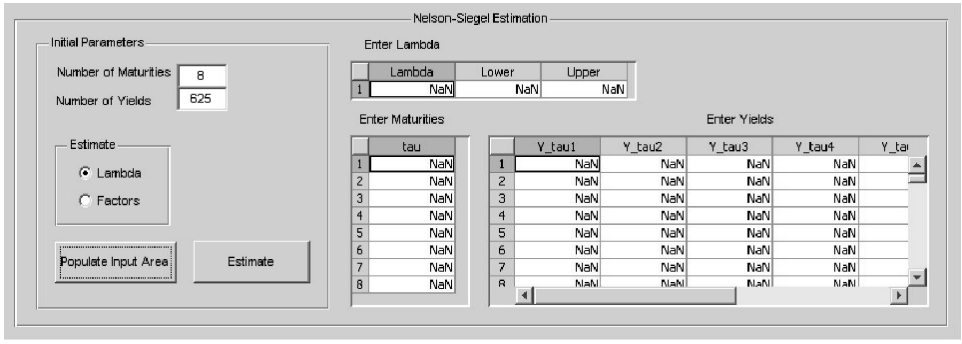


Figure 8.8: The expanded input area.

friendly GUIs. The only feature we still need to present is how to transfer values from a panel of "radio buttons" to the "handles" structure. This is shown below in the code that apply to the "Estimate button".

```
.....................................................................
[1] function Estimate_Callback(hObject, eventdata, handles)
[2] %
[3] handles = guidata(hObject);
[4] uitable_lambdas = str2double(cell(handles.uitable_lambdas.getData));
[5] uitable_maturities = str2double(cell(handles.uitable_maturities.getData));
[6] uitable_Yields     = str2double(cell(handles.uitable_Yields.getData));
[7] tst       = get(handles.Estimate_panel,'SelectedObject');
[8] Estimate_ = get(tst,'String');
[9] [nObs nMaturities] = size(uitable_Yields);
[10] if ( strcmp(Estimate_, 'Factors') )
[11]    [Beta,u,Res,optim ] = NS_gui_code(uitable_Yields, uitable_maturities, uitable_lambdas,
0);
[12]   %...
[13]   %...  Constructing Output
[14]   %...
[15]   %...  Betas
[16]   c_Beta=corr(Beta);
[17]   figure; plot(Beta); title('Estimates factors'); xlabel('Obs#'); ylabel('Value');

[18]   figure;
[19]   uitable(Beta, {'Level', 'Slope', 'Curvature'}, 'Position', [10 30 270 350]);
```

```
[20]   uicontrol('Style','text','String','Factors', 'Position', [70 385 80 15]);
[21]   uitable(c_Beta, {'Level', 'Slope', 'Curvature'}, 'Position', [300 310 250 70]);
[22]   uicontrol('Style','text','String','Correlation of factor', 'Position', [330 385
200 15]);
[23]   %...  residuals
[24]   c_u = corr(u);
[25]   m_u = mean(u);
[26]   figure; surf(u); title('Residuals'); xlabel('Maturity#'); ylabel('Obs#'); zlabel('Value');
[27]   for ( j=1:nMaturities )
[28]      colnames(1,j) = cellstr(strcat('Res_tau ', num2str(j)));
[29]   end
[30]   figure; uitable(u, colnames, 'Position', [50 30 450 350]);
[31]   uicontrol('Style','text','String','Residuals', 'Position', [180 385 80 15]);
[32]   figure; uitable(c_u, colnames, 'Position', [50 180 450 160]);
[33]   uicontrol('Style','text','String','Correlation of Residuals', 'Position', [150
345 280 15]);
[34]   figure; bar(m_u); title('Average Residuals'); xlabel('Maturity#');
[35]  else
[36]   [Beta,u,Res,optim] = NS_gui_code(uitable_Yields, uitable_maturities, uitable_lambdas,
1);
[37]   %...
[38]   %...  Constructing Output
[39]   %...
[40]   figure; plot(Res(:,2),Res(:,1)./nObs); title('Sum Resid^2'); xlabel('Lambda');
ylabel('Squared error');
[41]   optimal_L = cellstr(strcat('Optimal Lambda= ', num2str(Res(optim,2))));
[42]   uicontrol('Style','text','String',optimal_L, 'Position', [150 345 280 15]);
[43] end
```

In the interest of completeness we choose to show the whole function call for the "Estimate button" although many of the programming components included in the function are well-know by now. We will hence not give a detailed account for all programming lines and steps above but rather focus on the few issues that have not been clarified so far. It is also seen that the "Estimate button" function draws on an external function called `NS_gui_code`. This code uses components from previous chapters and we will leave it to the reader to investigate this function further. Here we will only state that it

contains two basic calculation functions: one that estimates the Nelson-Siegel $\lambda-$coefficient and one that estimates the Nelson-Siegel yield curve factors.

Line [7]-[8] show how to extract the choice made by the user in a panel of "radio buttons". In the above this is done first by constructing a temporary variable called "tst" which take the value of the radio button chosen by the user. This is done by the use of the function call `get(handles.Estimate_panel,'SelectedObject')`, where "handles.Estimate_panel" refers to the name of the defined "radio button panel" and "selected object" extracts the choice made by the user. Line [8] converts the temporary variable into "string" format and stores this string the the variable called "Estimate_" . It is now possible to use this variable further in the program as it is done for example in line [10], where the function "strcmp" is used to compare two strings in an effort to execute the code in a way such that the choice made by the user is fulfilled.

The remaining parts of the code represent familiar material, and we leave it to the user for further scrutinization.

Chapter 9

Useful Formulas and Expressions

9.1 Introduction

This section collects some useful formulas and expressions.

9.2 Matrix operations

9.2.1 Definitions

A collection of numbers in a two dimensional array is called a matrix. If the array has r rows and c columns the matrix A can be defined as:

$$A_{(rxc)} = \begin{bmatrix} a_{(1,1)} & a_{(1,2)} & \cdots & a_{(1,c)} \\ a_{(2,1)} & a_{(2,2)} & \cdots & a_{(2,c)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(r,1)} & a_{(r,2)} & \cdots & a_{(r,c)} \end{bmatrix} = \left[a_{(i,j)}\right], \ \forall i = \{1, \ldots, r\} \wedge j = \{1, \ldots, c\}$$

When r=1 the matrix is said to be a row vector and when c=1 the matrix is said to be a column vector. When r=c the matrix is said to be square. The collection of diagonal elements of a square matrix is denoted by:

$$diag\left[A_{(rxr)}\right] = \begin{bmatrix} a_{(1,1)} \\ a_{(2,2)} \\ \vdots \\ a_{(r,r)} \end{bmatrix} = \left[a_{(i,i)}\right],\ i = \{1, 2, \ldots, r\}$$

9.2.2 Sum

$$A_{(rxc)} + B_{(rxc)} = a_{(r,c)} + b_{(r,c)}$$

$$(A + B) + C = A + (B + C)$$

9.2.3 Product

$$A_{(rxc)} * B_{(cxq)} = C_{(rxq)}$$

$$A * B \neq B * A$$

9.2.4 Transpose

$$(A^T)^T = A$$

$$(A + B + C)^T = A^T + B^T + C^T$$

$$(A * B * C)^T = C^T * B^T * A^T$$

9.2.5 Symmetric matrix

When $A = A^T$ the matrix A is said to be symmetric.

9.2.6 The Identity matrix:

$$I_{(nxn)} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$A_{(r,c)} * I_{(c,c)} = A_{(r,c)}$$

$$A_{(c,r)} * I_{(r,r)} = A_{(c,r)}$$

9.2.7 Determinant

The determinant is only defined for square matrices. For two-dimentional matrices the determinant has a geometrical interpretation as the area of the parallelogram spanned by the two vectors. Generally, the determinant can be calculated recursively. Let $A_{(-1,-c)}$ be the matrix $A_{(cxc)}$ where the first row and the c'th column has been deleted, then:

$$\det(A) = \sum_{c=1}^{n} (-1)^{c+1} a_{(1,c)} \det\left[A_{(-1,-c)}\right]$$

In a 2x2 case this calculation yields:

$$\det\left(\begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}\right) = 1 * 4 * 3 - 1 * 2 * 1 = 10.$$

$$\det(A * B) = \det(A) * \det(B)$$

$$\det\left(A^{-1}\right) = 1/\det(A)$$

9.2.8 Rank

A matrix is said to have full rank if all columns (rows) are linearly independent. A matrix having rank $\rho < c$, where c is the number of columns in the matrix is said not to have full rank. Let c be the number of columns and r be the number of rows of a given matrix, then:

$$\begin{aligned} \rho\left[A_{(r,c)}\right] &\leq \min(r,c), \\ \rho(A) &= \rho\left(A^T\right) \\ \rho\left(A^T * A\right) &= \rho\left(A * A^T\right) = \rho(A) \end{aligned}$$

When $\rho(A) = r$ the matrix is said to have full row rank and when $\rho(A) = c$ it is said to have full column rank. The rank concept is important because it relates to the invertability of matrices. For a square matrix $S_{(c,c)}$ with rank c, S is said to be non-singular and a unique S^{-1}, the inverse of S, exists. When the rank of S is less than c, S is said to be singular and its inverse does not exist.

9.2.9 Inverse

If the determinant of a matrix $A_{(rxr)}$ is different from zero then the inverse of A exists and is denoted by A^{-1}.

$$(A * B * C)^{-1} = C^{-1} * B^{-1} * A^{-1}$$

$$\left(A^T\right)^{-1} = \left(A^{-1}\right)^T$$

$$A * A^{-1} = I$$

$$[\alpha A]^{-1} = \alpha^{-1} A^{-1}, \ for\ \alpha \neq 0$$

A non-singular matrix is one for which the inverse exists. A singular matrix is one which has a determinant of zero. Thus, the inverse of a singular matrix does not exist.

9.2.10 Trace

$$Tr(A) = \sum_j a_{jj}$$

$$Tr(A + B) = Tr(A) + Tr(B)$$

$$Tr(A * B) = Tr(B * A)$$

9.2.11 Powers

For a square matrix:

$$A^k = \prod_k A$$

$$A^0 = I$$

9.2.12 Eigenvalues and eigenvectors

For a square matrix A the eigenvalues and eigenvectors can be found as the solution to:

$$A * c = \lambda * I * c$$

which can be written as:

$$(A - \lambda * I) * c = 0,$$

where c is called eigen vector and λ the associated eigenvalue. From the above it can be deduced that for $c \neq 0$, it must be fulfilled that the matrix $(A - \lambda * I)$ is singular, since if $(A - \lambda * I)$ is non-singular we would get $c = 0$ by pre-multiplying the above with $(A - \lambda * I)^{-1}$. Hence, a solution to the eigenvalues emerges from:

$$\det(A - \lambda * I) = 0.$$

9.2.13 Positive definite

The symmetric matrix $A_{(r,r)}$ is said to be positive semidefinite if:

$$x^T * A * x \geq 0,$$

where $x_{(r,1)}$ is a vector. For the positive definite matrix $A_{(r,r)}$ and vector $x_{(r,1)}$ the following holds:

$$x^T * A * x > 0.$$

9.2.14 Matrix differentiation

For vectors a and b and matrix X the following rules apply:

$$\frac{\partial\left(a^T * b\right)}{\partial b} = \frac{\partial\left(b^T * a\right)}{\partial b} = a$$

$$\frac{\partial\left(b^T * X * b\right)}{\partial b} = 2 * X * b.$$

9.3 Decompositions

9.3.1 Triangular

$$A = L * D * L^T$$

is valid and unique for any symmetric positive definite matrix A. L is lower triangular and D is a diagonal matrix.

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{r1} & l_{r2} & l_{r3} & \cdots & 1 \end{bmatrix} \text{ and } D = \begin{bmatrix} d_{11} & 0 & 0 & \cdots & 0 \\ 0 & d_{22} & 0 & \cdots & 0 \\ 0 & 0 & d_{33} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & d_{rr} \end{bmatrix}.$$

9.3.2 Cholesky

$$A = P * P^T,$$

where $P = L * D^{1/2}$, using the notation from the above triangular factorisation. So, P is lower triangular as L, however its diagonal is populated by the square root of the elements of D.

$$P = \begin{bmatrix} \sqrt{d_{11}} & 0 & 0 & \cdots & 0 \\ l_{21}\sqrt{d_{11}} & \sqrt{d_{22}} & 0 & \cdots & 0 \\ l_{31}\sqrt{d_{11}} & l_{32}\sqrt{d_{22}} & \sqrt{d_{33}} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{r1}\sqrt{d_{11}} & l_{r2}\sqrt{d_{22}} & l_{r3}\sqrt{d_{33}} & \cdots & \sqrt{d_{rr}} \end{bmatrix}.$$

9.3.3 Eigenvalue

$$A = C * \Lambda * C^{-1},$$

is the eigenvalue decomposition when A has eigenvalues that are different from each other. Λ collects the eigenvalues along the diagonal and C collects the eigenvectors.

$$\Lambda_{(r,r)} = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & \lambda_r \end{bmatrix} \text{ and } C_{(r,r)} = \begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_r \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}.$$

Since the eigenvalues are distinct the determinant of C exists and so does then C^{-1}. Note that:

$$A = C * \Lambda * C^{-1} \Leftrightarrow A * C = C * \Lambda,$$

9.4 Basic rules

9.4.1 Index rules

$a^m * a^n = a^{m+n}$	Equal bases
$a^m / a^n = a^{m-n}$	
$a^m * b^m = (a * b)^m$	Equal indices
$a^m / b^m = (a/b)^m$	
$a^{-n} = 1/a^n$	
$(a^m)^n = a^{m*n}$	Power of a power rule
$a^0 = 1$	
$\sqrt[m]{a} = a^{1/m}$	Square root
$\sqrt[m]{a^n} = (\sqrt[m]{a})^n = a^{n/m}$	

9.4.2 Logarithm rules

$$\begin{array}{l}\ln(e) = 1\\ \ln(1) = 0\\ \ln(e^m) = m\\ \ln(a*b) = \ln(a) + \ln(b)\\ \ln(a/b) = \ln(a) - \ln(b)\\ \ln(a^m) = m*\ln(a)\\ \ln(\sqrt[m]{a}) = 1/m*\ln(a)\end{array}$$

9.4.3 Simple derivatives

$f(x)$		$f'(x)$
c	$\rightarrow$	0
$\ln(x)$	$\rightarrow$	$\frac{1}{x}$
e^x	$\rightarrow$	e^x
e^{kx}	$\rightarrow$	ke^{kx}
a^x	$\rightarrow$	$a^x \ln(a)$
x^a	$\rightarrow$	ax^{a-1}
$\frac{1}{x}\left(=x^{-1}\right)$	$\rightarrow$	$-\frac{1}{x^2}\left(=-x^{-2}\right)$
$\sqrt{x}\left(=x^{1/2}\right)$	$\rightarrow$	$\frac{1}{2\sqrt{x}}\left(=\frac{1}{2}x^{-1/2}\right)$
$\sin(x)$	$\rightarrow$	$\cos(x)$
$\cos(x)$	$\rightarrow$	$-\sin(x)$

Definition

$$f'(x) = \frac{df}{dx} = \lim_{h\to 0}\frac{f(x+h) - f(x)}{h}$$

Linearity

$$\frac{d}{dx}\left[f(x) + g(x)\right] = \frac{df(x)}{dx} + \frac{dg(x)}{dx},$$

$$\frac{d}{dx}\left[c*f(x)\right] = c*\frac{df(x)}{dx}.$$

Product rule

$$\frac{d}{dx}\left[f(x)*g(x)\right] = f'(x)*g(x) + f(x)*g'(x).$$

Quotient rule

$$\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x) * g(x) - f(x) * g'(x)}{[g(x)]^2}.$$

Chain rule

$$\frac{d}{d(x)} = [f(g(x))] = f'[g(x)] * g'(x) = \frac{df}{dg} * \frac{dg}{dx}.$$

9.4.4 Simple integrals

$f(x)$		$F(x)$
0	$\rightarrow$	k
k	$\rightarrow$	kx
$\ln(x)$	$\rightarrow$	$x\ln(x) - x$
e^x	$\rightarrow$	e^x
e^{kx}	$\rightarrow$	$\frac{1}{k}e^{kx}$
a^x	$\rightarrow$	$\frac{a^x}{\ln(x)}$
x^a	$\rightarrow$	$\frac{1}{a+1}x^{a+1}$
$\frac{1}{x}\left(= x^{-1}\right)$	$\rightarrow$	$\ln\lvert x\rvert$
$\sqrt{x}\left(= x^{1/2}\right)$	$\rightarrow$	$\frac{2}{3}x\sqrt{x}\left(= \frac{2}{3}x^{3/2}\right)$
$\sin(x)$	$\rightarrow$	$-\cos(x)$
$\cos(x)$	$\rightarrow$	$\sin(x)$

Indefinite integral

$$\int f(x)\,dx = F(x) + c.$$

Integration is linear

$$\int [f(x) + g(x)]\,dx = \int f(x)\,dx + \int g(x)\,dx,$$

if c is a constant

$$\int cf(x)\,dx = c\int f(x)\,dx.$$

Definite integral

$$\int_a^b f(x)\,dx = [F(x)]_a^b = F(b) - F(a),$$

$$\int_a^a f(x)\,dx = 0,$$

$$\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx,$$

$$\int_a^b f(x)\,dx + \int_b^c f(x)\,dx = \int_a^c f(x)\,dx.$$

9.5 Distributions

9.5.1 Normal

$$f_{Y_t}(y_t) = \frac{1}{2\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(y_t-\mu)^2}{2\sigma^2}\right],$$

the variable Y_t has a normal distribution with mean μ and variance σ^2, i.e. $Y_t \sim N(\mu, \sigma^2)$.

9.5.2 Multivariate normal

$$f_Y(y) = (2\pi)^{-n/2} |\Omega|^{-1/2} \exp\left[(-1/2)(y-\mu)'\Omega^{-1}(y-\mu)\right],$$

with $E\left[(Y-\mu)(Y-\mu)'\right] = \Omega$. Note that $|\Omega|^{-1/2} = |\Omega^{-1}|^{1/2}$.

9.5.3 t-distribution

[to be inserted]

9.5.4 Copulas

[to be inserted]

9.5.5 Vasicek's limiting distribution

This density is relevant in credit risk applications and gives the distribution of losses due to default based on the following inputs: ρ is the correlation between issuers/bonds and p is the default probability. It is assumed that ρ and p are identical across all issuers/bonds. The density gives the limiting loss distribution for a portfolio with k issuers/bonds where $k \to \infty$.

$$f\left(x;p,\rho\right)=\sqrt{\frac{1-\rho}{\rho}}\exp\left[-\frac{1}{2\rho}\left(\sqrt{1-\rho}N^{-1}\left(x\right)-N^{-1}\left(p\right)\right)^{2}+\frac{1}{2}\left[N^{-1}\left(x\right)\right]^{2}\right],$$

where N^{-1} is the inverse normal distribution. The cumulative distribution function is given by:

$$F\left(x;p,\rho\right)=N\left(\frac{\sqrt{1-\rho}N^{-1}\left(x\right)-N^{-1}\left(p\right)}{\sqrt{\rho}}\right).$$

The loss distribution is defined on the interval $0 \leq x \leq 1$, so x can be interpreted as the fraction of the portfolio that is lost.

9.6 Functions

9.6.1 Linear (affine) function

$$f\left(x\right)=c+b*x$$

where c and b are scalar constants. c is the function value when $x=0$ and hence where the function crosses the y-axis; this is also referred to as the intercept. b is the slope of the function i.e. how much it responds to the variable x. In a matrix context the linear equation of many variables can be expresses in a compact form as:

$$y_{(r,1)}=c+\begin{bmatrix} x_{1,t} & x_{2,t} & \cdots & x_{n,t} \\ x_{1,t-1} & x_{2,t-1} & \cdots & x_{n,t-1} \\ \vdots & \vdots & & \vdots \\ x_{1,1} & x_{2,1} & \cdots & x_{n,1} \end{bmatrix}*\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}=c_{(1,1)}+X_{(r,c)}*b_{(c,1)},$$

where $x_1, x_2, ...$ represent different variables and the vector b collects the slope coefficients for each variable.

9.6.2 Quadratic function

$$f(x) = c + b * x + a * x^2$$

where c, b and a are scalar constants. The solutions:

$$if\ b^2 \geq 4 * a * c \iff x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 * a * c}}{2 * a}$$

if $b^2 < 4 * a * c$ then the quadratic equation has no real roots (but a solution with complex roots exists). The function has a maximum point if $a < 0$ i.e. then it is $\cap-$shaped, and a minimum point if $a > 0$ i.e. then it is $\cup-$shaped. This minimum or maximum point is defined by:

$$t_{1,2} = \left(-\frac{b}{2 * a}, \frac{4 * a * c - b^2}{4 * a} \right).$$

9.6.3 General polynominals

$$f(x) = a_n * x^n + a_{n-1} * x^{n-1} + \ldots + a * x + a = 0$$

is a polynominal of order n.

9.6.4 Exponential

$$f(x) = a^x$$

is a general exponential function with base a. Using the natural base $e \approx 2.718$ gives the natural exponential function:

$$f(x) = \exp(x) = e^x.$$

9.6.5 Logarithm

$$f(x) = \ln(x),$$

$$y = \ln(x) \iff x = e^{y.}$$

9.6.6 Error function

$$\text{erf}\left(x\right)=\frac{2}{\sqrt{\pi}}\int_{0}^{x}e^{-u^{2}}du$$

is the integral of the standard normal distribution.

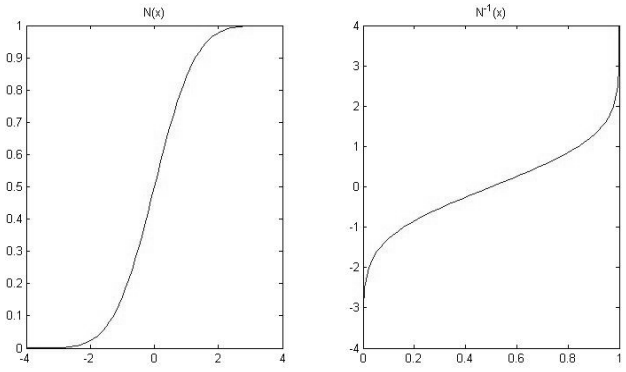
9.6.7 Inverse

$$f^{-1}\left(x\right)$$

is said to be the inverse function of $f\left(x\right)$. It is important to note that $f^{-1}\left(x\right)$ does **not** mean $1/f\left(x\right)$ i.e. " -1" does not refer to the power of the function as is otherwise customary. Some examples are:

$f\left(x\right)$	$f^{-1}\left(x\right)$
$\ln\left(x\right)$	$\exp\left(x\right)$
$x^{a/b}$	$x^{b/a}$
x^{a}	$x^{1/a}$

A graphical illustration is shown below for the cumulative normal distribution and its inverse:



An example of a function and its inverse

9.7 Taylor series approximation

$$f\left(x+\Delta\right)=f\left(x\right)+\frac{df}{dx}*\Delta+\frac{1}{2!}\frac{d^{2}f}{dx^{2}}*\Delta^{2}+\frac{1}{3!}\frac{d^{3}f}{dx^{3}}*\Delta^{3}+\frac{1}{r!}\frac{d^{r}f}{dx^{r}}*\Delta^{r}+R,$$

$r!$ is the factorial of r i.e. $r*\left(r-1\right)*\left(r-2\right)*\ldots*2*1$.

9.8 Interest rates, returns and portfolio statistics

The formulas and expressions given below shows how to calculate interest rates and returns over multiple periods. It is also shown how summary risk measures like modified duration can be aggregated at the portfolio level.

9.8.1 Cummulative arithmetic return

$$r_{1,n} = r_1 + r_2 + \ldots + r_n = \sum_{j=1}^{n} r_j$$

9.8.2 Average arithmetic return

$$\overline{r} = \frac{1}{n} * (r_1 + r_2 + \ldots + r_n) = \frac{1}{n} * \sum_{j=1}^{n} r_j$$

9.8.3 Cummulative geometric return

$$r_{1,n} = [(1 + r_1) * (1 + r_2) * \ldots * (1 + r_n)] - 1 = \prod_{j=1}^{n} (1 + r_j) - 1$$

if the interest rate in each period is the same then the expression becomes:

$$r_{1,n} = (1 + r)^n - 1$$

9.8.4 Average geometric return

$$\overline{r} = [(1 + r_1) * (1 + r_2) * \ldots * (1 + r_n)]^{1/n} - 1 = \left[\prod_{j=1}^{n} (1 + r_j)\right]^{1/n} - 1$$

9.8.5 Compounding of interest rates

Let m be the compounding interval, e.g. $m = 12$ means that the annual rate of r is paid on a monthly basis, then the effective interest rate is:

$$r^{eff} = \left[1 + \left(\frac{r}{m}\right)\right]^m - 1.$$

In the case of continously compounded interest rate, i.e. when $m \to \infty$ the effective rate can be calculated by:

$$r^{eff} = e^r - 1$$

and for multiple periods this becomes:

$$r_{1,n}^{eff} = e^{r*n} - 1.$$

9.8.6 Portfolio statistics

Let w be the vector of amounts invested in each of the j securities covered by the eligible investment universe. Let r be the vector of expected returns, C the expected covariance matrix, MD the vector of modified durations and $Conv$ the vector of convexities. Then, letting p denote portfolio level and σ^2 the variance, the following applies:

$$r_p = w^T * r$$

$$\sigma_p^2 = w^T * C * w$$

$$MD_p = w^T * MD$$

$$Conv_p = w^T * Conv$$

Bibliography

Ang, A., and M. Piazzesi (2003): "A No-Arbitrage Vector Autoregression of Term Structure Dynamics with Macroeconomic and Latent Variables," *Journal of Monetary Economics*, 50(4), 745–787.

Ang, A., M. Piazzesi, and M. Wei (2006): "What Does the Yield Curve Tell us about GDP Growth," *Journal of Econometrics*, 131, 359–403.

Bjork, T. (2004): *Arbitrage Theory in Continuous Time*. Oxford University Press, Oxford.

Brandimarte, P. (2002): *Numerical Methods in Finance*. Wiley, New York.

Brigo, D., and F. Mercurio (2001): *Interest Rate Models, Theory and Practice*. Springer Finance, Berlin.

Cairns, A. J. G. (2004): *Interest Rate Models - An Introduction*. Princeton University Press, Princeton, New Jersey.

Campbell, J. W., A. W. Lo, and A. C. MacKinlay (1997): *The Econometrics of Financial Markets*. Princeton University Press, Princeton, New Jersey.

Chen, R., and L. Scott (1993): "Maximum likelihood estimation for a multi-factor equilibrium model of the term structure of interest rates," *Journal of Fixed Income*, 3, 14–31.

Christensen, J., F. Diebold, and G. Rudebusch (2007): "The affine arbitrage-free class of nelson-siegel term structure models," *Federal Reserve Bank of San Francisco and University of Pennsylvania*.

Coroneo, L., K. Nyholm, and R. Vidova-Koleva (2007): "How Arbitrage-Free is the Nelson-Siegel Model," *Risk Management Division, European Central Bank*.

Cox, J., J. Ingersoll, and S. Ross (1985): "A Theory of the Term Structure of Interest Rates," *Econometrica*, 53, 385–407.

Dai, Q., and K. Singleton (2000): "Specification analysis of affine term structure models," *Journal of Finance*, 55, 1943–1978.

Diebold, F., and C. Li (2006): "Forecasting the Term Structure of Government Bond Yields," *Journal of Econometrics*, 130, 337–364.

Down, K. (2005): *Measuring Market Risk*. Wiley Finance, Chichester, West Sussex, PO19 8SQ, England.

Duffie, D., and R. Kan (1996): "A yield-factor model of interest rates," *Mathematical Finance*, 6(379-406).

Glasserman, P. (2004): *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York.

Greene, W. H. (1993): *Econometric Analysis*. Macmillian Publishing Company.

Hamilton, J. D. (1994): *Time Series Analysis*. Princeton University Press, Princeton, New Jersey.

Huang, C., and R. H. Litzenberger (1988): *Foundations for Financial Economics*. Prentice-Hall.

Jackel, P. (2002): *Monte Carlo Methods in Finance*. Wiley, West Sussex, England.

James, J., and N. Webber (2000): *Interest Rate Modelling*. John Wiley and Sons, West Sussex, England.

Johnson, R. A., and D. W. Wichern (1992): *Applied Multivariate Statistical Analysis*. Prentice-Hall, London.

Kim, C., and C. Nelson (2000): *State Space Models with Regime Switches*. MIT press.

Lipschutz, S., and M. L. Lipson (2001): *Schaums outline of linear algebra*. McGraw-Hill, New York.

Litterman, R., and J. Scheinkman (1991): "Common Factors Affecting Bond Returns," *Journal of Fixed Income*, June, 54–61.

Luenberger, D. G. (1998): *Investment Science*. Oxford University Press, New York.

Lund, J. (1998): *Review of Continuous-Time Term-Structure Models*. www.jesperlund.com.

McLeish, D. L. (2005): *Monte Carlo Simulation and Finance*. Wiley, New Jersey.

Merton, R. (1973): "Theory of Rational Option Pricing," *Bell Journal of Economics and Management Science*, 4, 141–183.

Meucci, A. (2005): *Risk and Asset Allocation*. Springer Finance, Springer-Verlag Berlin Heidelberg New York.

Mills, T. C. (1999): *The Econometric Modelling of Financial Time Series*. Cambridge University Press, Cambridge.

Mönch, E. (2006): "Forecasting the Yield Curve in a Data-Rich Environment: A No-Arbitrage Factor-Augmented VAR Approach," ECB Working Paper series No. 544.

Nelson, C. R., and A. F. Siegel (1987): "Parsimonious Modeling of Yield Curves," *Journal of Business*, 60, 473–489.

Rebonato, R. (1998): *Interest-Rate Option Models*. John Wiley and Sons, West Sussex, England.

Söderlind, P., and L. Svensson (1997): "New Techniques to Extract Market Expectations from Financial Instruments," *Journal of Monetary Economics*, 40, 383–429.

Taylor, J. (1993): "Discretion versus Policy Rules in Practice," *Carnegie-Rochester Conference Series on Public Policy*, 39, 195–214.

Tuckman, B. (2002): *Fixed Income Securities*. Wiley, New Jersey.

VASICEK, O. (1977): "An Equilibrium Characterization of the Term Structure," *Journal of Financial Economics*, 5, 177–188.