MODELLING MARKET RISK WITH SAS RISK DIMENSIONS:

A STEP BY STEP IMPLEMENTATION

CARL DU TOIT

Modelling Market Risk with SAS Risk Dimensions:

A Step By Step Implementation

Carl du Toit

Assignment presented in partial fulfillment of the requirements for the degree of

MASTER OF COMMERCE

in the Department of Statistics and Actuarial Science,

Faculty of Economic and Management Sciences,

University of Stellenbosch

Supervisor:

Prof. W.J. Conradie

April 2005

DECLARATION

I, the undersigned, hereby declare that the work contained in this assignment is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: ____________________

Date: _____________________

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to:

- my supervisor, **Professor Willie Conradie**. I thank you for your time, encouragement, suggestions and contributions in the preparation of this document. Thank you for the role that you have played in this regard.
- **Professor Michiel Kruger** from the North West University, for first introducing me to SAS Risk Dimensions. Thank you very much for all the technical assistance, and for sharing your invaluable knowledge of the SAS software package. Your input is much appreciated.
- **Suzanne Steyn**, for tending to the grammar of the text in the document.
- the **Department of Statistics and Actuarial Science** at the University of Stellenbosch, for accepting me as a postgraduate student.
- my **family** for all their support. I thank my mother, brother, sister, brother in law and all the other family members for their interest, encouragement and support throughout my study period. I would also like to extend a special word of thanks to my late father. He has played a huge role in my life and education.
- My girlfriend **Carina**. I thank you for your moral support and love.
- My friend **Dewald**. Thank you for being a great study partner and friend. You have motivated, encouraged and guided me a lot during the last few years.
- All my other **friends** that supported me throughout the preparation of this document.
- **God**, who has provided me with talents, always supported me and blessed me with wonderful family and friends.

SUMMARY

Financial institutions invest in financial securities like equities, options and government bonds. Two measures, namely return and risk, are associated with each investment position. Return is a measure of the profit or loss of the investment, whilst risk is defined as the uncertainty about return.

A financial institution that holds a portfolio of securities is exposed to different types of risk. The most well-known types are market, credit, liquidity, operational and legal risk. An institution has the need to quantify for each type of risk, the extent of its exposure. Currently, standard risk measures that aim to quantify risk only exist for market and credit risk. Extensive calculations are usually required to obtain values for risk measures. The investments positions that form the portfolio, as well as the market information that are used in the risk measure calculations, change during each trading day. Hence, the financial institution needs a business tool that has the ability to calculate various standard risk measures for dynamic market and position data at the end of each trading day.

SAS Risk Dimensions is a software package that provides a solution to the calculation problem. A risk management system is created with this package and is used to calculate all the relevant risk measures on a daily basis.

The purpose of this document is to explain and illustrate all the steps that should be followed to create a suitable risk management system with SAS Risk Dimensions.

OPSOMMING

Finansiële instellings belê weens die aard van hul sakebedrywighede in finansiële instrumente soos aandele, opsies, termynkontrakte en staatseffekte. Twee maatstawwe, naamlik opbrengs en risiko word gekoppel aan elke beleggingsposisie wat in ’n finansiële instrument geneem word. Die wins of verlies van die belegging word gemeet deur die opbrengs, terwyl risiko die onsekerheid ten opsigte van die opbrengs verteenwoordig.

’n Finansiële instelling wat oor ’n portefeulje van beleggings beskik, is blootgestel aan verskeie soorte risiko’s, naamlik mark-, krediet-, likiditeits-, operasionele- en wetlike risiko. Die instelling wil graag sy blootstelling aan elke een van hierdie tipes risiko kwantifiseer. Daar bestaan tans slegs vir mark- en kredietrisiko, standaard risikomaatstawwe wat in die industrie algemeen aanvaar en gebruik word. Die berekening van die risikomaatstawwe is gewoonlik baie rekenintensief. Markinligting sowel as inligting oor elke beleggingsposisie in die huidige portefeulje word gebruik in bogenoemde berekeninge. Hierdie inligting verander egter gedurende elke verhandelingsdag. Die finansiële instelling benodig dus programmatuur om - byvoorbeeld aan die einde van elke verhandelingsdag - sekere standaard risikomaatstawwe te bereken op grond van die nuutste mark- en beleggingsdata.

Die gebruik van SAS Risk Dimensions bied ’n oplossing vir die berekeningsprobleem. ’n Risikobestuur stelsel waarmee die verlangde risiko maatstawwe daagliks bereken word, kan deur middel van hierdie sagteware pakket geskep word.

In hierdie dokument verduidelik en illustreer ons die stappe wat gevolg moet word om ‘n gepaste risikobestuur stelsel in SAS Risk Dimensions, te implementeer.

CONTENTS

1

INTRODUCTION AND OVERVIEW

1.1 Introduction

The core business of many financial institutions, is to invest in financial instruments like equities, government bonds, foreign currency, interest rate swaps, options, futures and lately more and more exotic instruments.

Financial institutions need to realize growth in the value of their assets in order to meet future liabilities. One example is life insurers that need to preserve their profits of today to provide for large expenditures in the future. Other institutions use financial instruments to remove uncertainty in their business environment. An example of this is airline companies that buy futures on jet fuel and effectively fix the price that they would have to pay for jet fuel in six months time, today.

For each investment decision there is an associated return and risk. **Return** is a measure of the profit/loss associated with the investment. **Risk** is defined as the uncertainty about the return of the investment. Expected return and risk are positively related to each other. Investors want compensation in the form of a larger expected return, when taking on more risk.

The most important kinds of risk are market risk, credit risk, operational risk, liquidity risk and legal risk. **Market risk** is the risk that the portfolio of financial instruments will decline in value, due to a change in market variables such as interest rates, exchange rates and equity prices. **Credit risk** is defined as the impact on the portfolio value when a counter party fails to perform an obligation. **Operational risk** is the risk of losing money due to operational failure. Power failures, the collapse of IT systems, staff problems (illness of key personnel, strikes etc.), the evacuation of the working place and other problems that may lead to operational failure. The risk of losing money when financial contracts are not enforceable is called **legal risk**. The fifth type of risk, namely **liquidity risk**, is the risk of losing money due to financial costs that may arise with liquidating a position held. The costs are determined by the relative liquidity or illiquidity of the market.

Various risk measures that aim to quantify market and credit risk, exist in practise. Examples are Value at Risk (VaR) and Credit Value at Risk (CVaR). These measures are used world-wide and are accepted standard risk measures. It is, however, more difficult to accurately quantify operational, liquidity and legal risk. Standard risk measures for these types of risk do not exist at present.

It is important for financial institutions to calculate all the available risk measures for the portfolio of financial instruments that are held. The goal is to use the information gathered from the risk measures to the company's advantage in subsequent investment decisions. The constituents of the portfolio, as well as, the relevant market information may change during each trading day. Thus, the need exists to calculate the available risk measures at the end of each trading day for the latest market and portfolio information. The results of the analyses need to be presented in a report that is easily interpretable. The risk managers of the financial institutions may use this information contained in the report, during the next trading day to make adjustments to the portfolio. Risk management systems are used to calculate the daily measures and to generate the required reports.

Risk management systems may be designed in **SAS Risk Dimensions.** The systems can calculate market and credit risk measures for even the most complex portfolios. Some of the available measures or analyses are Value at Risk (VaR), sensitivity analysis, scenario analysis, stress testing, current exposure analysis, potential exposure analysis, credit rating migration analysis, descriptive statistics, cash flow analysis and portfolio optimization analysis. Monte Carlo simulation is used during some of these analyses. SAS Risk Dimensions also offers an extensive reporting system. The execution of the risk management system generates reports that present all the necessary risk measures in a user-defined way. SAS Risk Dimensions is a powerful business tool that is more than capable of updating the risk measures on a daily basis.

Hence, the purpose of this document is to explain and illustrate the steps that are necessary to create a suitable risk management system in SAS Risk Dimensions. The document will make it easier for people with a risk management background to understand and use SAS Risk Dimensions.

Not all the features of SAS Risk Dimensions are discussed in detail in this document. These features are usually more advanced and are used for a specific business need. It is neither less important or useful.

1.2 An overview of the document

The portfolio of financial instruments that is held by a fictitious company named ***Activegrowth Limited***, is used throughout the document. This case study is discussed in detail in Section 3.1. The company invests in five financial instruments, namely equities, options, futures, government bonds and interest rate swaps. The company needs to calculate amongst other risk measures, a Value at Risk estimate for the portfolio that is currently held.

The first step in the implementation of a suitable risk management system is the creation of data files that contain the relevant market and position information of the company.

Consider the following extracts from the three trade books of the company, that contain the relevant position information:

InstType	**Instid**	**Short**	**Holding**	**Premium**	**Sector**	**Strike**	**Enddate**	**Opttype**	**Cprice.**
Equity	SOL_001	0	400	94.7	Res	.	.	.	.
Equity	SLM_002	1	8500	7.8	Fin	.	.	.	.
Equity	ASA_001	0	2800	30	Fin	.	.	.	.
Equity	ASA_002	0	2000	28.5	Fin	.	.	.	.
Equity	OML_001	0	8200	12.5	Fin	.	.	.	.
Equity	OML_002	1	1400	10.1	Fin	.	.	.	.
Future	ASA_QM4	0	10000	.	.	.	17-Jun-04	.	46.59
Future	OML_Q43	0	15000	.	.	.	17-Jun-04	.	11.93
Future	SLM_Q42	0	4000	.	.	.	17-Jun-04	.	9.54
Future	SOL_Q41	1	14000	.	.	.	17-Jun-04	.	97.86
Future	SOL_Q42	0	12000	.	.	.	17-Jun-04	.	103.29
Option	ASA_O02	0	6000	4.67	.	40	29-Jun-04	EC	.
Option	ASA_O06	0	10000	3.8	.	33	14-Sep-04	EP	.
Option	SOL_O04	1	5000	5	.	88	18-Oct-04	EC	.
Option	SOL_O05	0	6000	4.3	.	93	27-Jul-04	EP	.
Option	SLM_O05	0	18000	1.2	.	8.8	15-Aug-04	EC	.

InstType	**Instid**	**Short**	**Notional**	**MaturityDate**	**Fromdate**	**Rcvetype**	**FixRate**	**Ftr_name**
Int_Swap	DB_IS_01	0	150000	12/17/2006	12/17/2003	Floating	0.06	JB_6_MTH
Int_Swap	IB_IS_02	0	1000000	4/17/2007	4/17/2004	Fixed	0.1	JB_6_MTH
Int_Swap	IB_IS_03	0	850000	5/17/2005	11/17/2003	Floating	0.065	JB_6_MTH

InstType	**Instid**	**Notional**	**Holding**	**MaturityDate**	**Coupon**	**Premium**	**Red_Amount**
Gov_Bond	R153_1	100	100	8/31/2010	0.13	85	100
Gov_Bond	R153_2	100	600	8/31/2010	0.13	84.3	100
Gov_Bond	R133_1	100	2400	9/15/2007	0.15	70	100
Gov_Bond	R177_1	100	2500	5/15/2007	0.095	98	100

Each row represents a position held in a financial instrument. The name of each instrument is included in the *Insttype* column. Other information about each position is included in the remaining columns.

Consider the following market information at the close of the last trading day, 13 May 2004:

Date	ASA	OML	SLM	SOL	Vol_ASA	Vol_OML	Vol_SLM	Vol_SOL	JB_6_MTH
05/13/2004	45	11.5	8.55	99	0.210693	0.247062	0.224054	0.28974	0.08303

The Absa equity price is included in the *ASA* column, for example, whilst the annualized volatility estimate of this equity price is included in the *Vol_ASA* column.

This position and market information are used, together with, other information in various Risk Dimensions structures to value each position in the portfolio and to calculate various risk measures, for example, Value at Risk. The calculated risk measures are presented in reports that are viewed in Output 1.1. The creation of these reports is the ultimate goal of the implementation of Risk Dimensions.

<u>Output 1.1: Reports</u>

```
      Market Report: 13 May 2004
       Portfolio Summary

     Type
      of          Mark to Market
  instrument        Value (ZAR)
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

    Equity            841,135.00
    Future            -71,826.63
   Gov_Bond           622,845.32
   Int_Swap            49,140.23
    Option              6,905.98
                 ===============
                    1,448,199.90
```

<u>Output 1.1 continues …</u>

```
                              Market report: 13 May 2004
                               95% 1-day Value at Risk

                                      Mark
                                       to                      VaR as
                    Instrument       Market      Value at    percentage  Estimated
Simulation method      Type          (ZAR)         Risk        of MtM    shortfall
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

  1. Hist_Sim: 1.      Equity       841,135.00    187,900.00       22.34  220,137.85
                      Future       -71,826.63     37,992.13       52.89   47,001.85
                     Gov_Bond      622,845.32      2,439.06        0.39    2,884.32
                     Int_Swap       49,140.23      2,303.78        4.69    2,674.56
                      Option         6,905.98     21,099.33      305.52   26,366.75
                        +        1,448,199.90    197,437.51       13.63  228,920.24
                                 ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ

   2. Cov_Sim: 1.       Equity       841,135.00     92,554.44       11.00  116,546.93
                      Future       -71,826.63     85,419.13      118.92  104,859.00
                     Gov_Bond      622,845.32     12,722.85        2.04   15,960.04
                     Int_Swap       49,140.23     10,632.96       21.64   12,902.86
                      Option         6,905.98     49,742.82      720.29   62,974.89
                        +        1,448,199.90    163,646.74       11.30  201,310.00
                                 ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ

  3. Model_Sim: 1       Equity       841,135.00     14,986.77        1.76   18,833.02
                      Future       -71,826.63     34,538.72       48.75   45,271.33
                     Gov_Bond      622,845.32      2,120.80        0.34    2,729.93
                     Int_Swap       49,140.23      1,994.84        4.06    2,480.13
                      Option         6,905.98      4,756.16       61.46   10,468.18
                        +        1,448,199.90     39,753.15        2.83   57,963.19
                                 ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ
```

The mark-to-market value, as well as, three Value at Risk estimates, are calculated each for the whole portfolio and five sub-portfolios. Every sub-portfolio consists of positions in financial instruments that are the same, for example, a sub-portfolio that consists only of positions in equities. Three methods are used to calculate value at risk, namely historical simulation, covariance-based Monte Carlo simulation and model-based Monte Carlo simulation. The results of these three methods are viewed in *Hist_Sim*, *Cov_Sim* and *Model_Sim* in Output 1.1.

The starting point of the risk management system is the creation of data files as viewed above. The end point of the risk management system is the creation of reports as viewed in Output 1.1. These steps, as well as the other steps that are necessary to implement a successful risk management system, are discussed in detail from Chapters 2 to 11.

The layout and working of the **SAS window environment** is discussed in Chapter 2. Basic SAS structures like SAS data sets, SAS programs and SAS libraries are used in this environment.

A number of **preparation steps** are needed before Risk Dimensions is activated. Various folders and data files that contain market, position and other information are created outside the SAS environment. SAS libraries are created inside the SAS environment. The data files that contain the market, position and other information are imported into SAS data sets. In the SAS window environment the SAS data sets are grouped in SAS libraries. The preparation steps that are necessary are discussed in Chapter 3.

SAS Risk Dimensions is activated and **risk environment(s)** are created in Chapter 4. A risk environment is defined as a collection of information and files, created to implement a risk management system.

The registration of **variables** in a risk environment is discussed in Chapter 5. The variables are used during the valuation of financial instruments and the calculation of risk measures or analyses.

Data preparation in the SAS environment is discussed in Chapter 6. The modification and combination of SAS data sets are used in this process. An alternative variable registration method, namely **data-driven registration** is also discussed in this chapter.

Special blocks of program code, called **method programs** are discussed in Chapter 7. Various kinds of method programs exist. Some method programs use variables that refer to market and position information to calculate data values for new variables. All these variables are used in other method programs to calculate the value of each instrument in the portfolio. For each real-life financial instrument in the portfolio, for example, a future or an equity, a corresponding

Risk Dimensions structure, namely an **instrument type** is defined within the risk environment. Instrument types are also discussed in Chapter 7.

The creation of statistical models called **risk factor models** is discussed in Chapter 8. The models are used to predict the future values of market variables such as interest rates and equity prices.

SAS data sets that contain **market information** that are used in the risk management system are registered in a risk environment in Chapter 9. Other SAS data sets that contain portfolio information are used to create a **portfolio file** in a risk environment. The portfolio file is further used in the risk management system. This is also discussed in Chapter 9.

Various risk analyses such as sensitivity analyses, profit/loss curves and scenario analyses are created in Chapter 10. An important Risk Dimensions structure, namely, a **project** is also created. The project combines the portfolio file, the method programs, the market information, risk analyses and other Risk Dimensions structures. The execution of a project leads to the calculation of the portfolio value and the risk analyses. The information that is created by the execution, is stored in SAS data sets called output data sets. Graphical illustrations of some of the risk analysis results are also created in the graphical user interface (GUI).

Reports are used to present the calculated risk measures or analyses in an easily interpretable and user-friendly way (see Output 1.1). The reports obtain the necessary information from the output data sets and are discussed in Chapter 11.

2

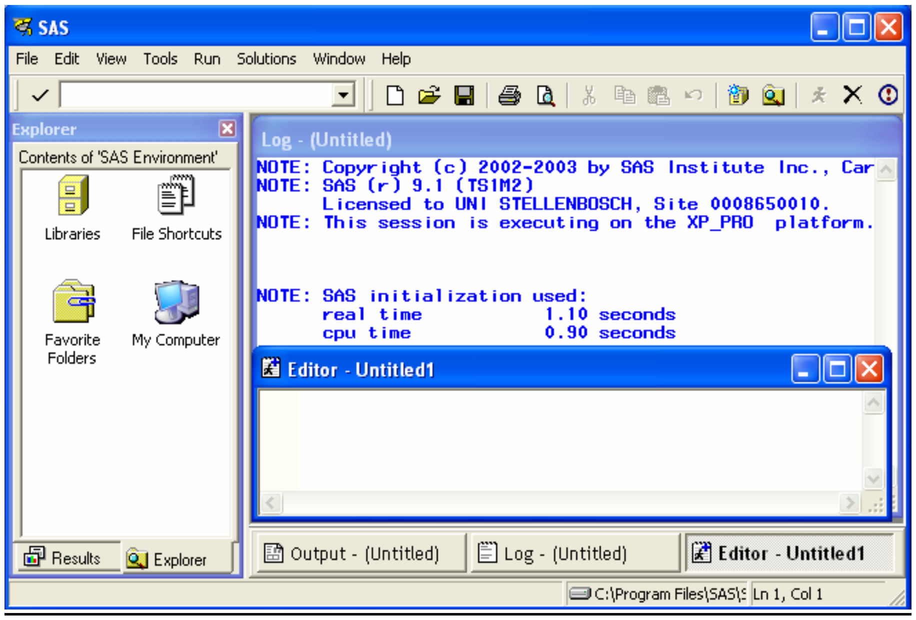
AN OVERVIEW OF THE SAS ENVIRONMENT

The SAS software package is activated by clicking on the Microsoft Windows desktop icon by name *SAS V9.* The package opens into an initial window called the **SAS window environment**. The SAS window environment consists of five different windows. Each window serves a different function in the environment. Important SAS structures like SAS programs, SAS libraries and SAS data sets are created in these windows. The creation and use of these SAS structures are essential in the implementation of a successful risk management system.

The purpose of this chapter is to discuss and explain the role that each window and SAS structure plays in the SAS window environment. The concepts mentioned above are illustrated in Example 2.1.

2.1 The SAS Window Environment

The SAS window environment, as illustrated in Figure 2.1 opens when the *SAS V9* icon is clicked on the desktop.



```
NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Car
NOTE: SAS (r) 9.1 (TS1M2)
      Licensed to UNI STELLENBOSCH, Site 0008650010.
NOTE: This session is executing on the XP_PRO  platform.



NOTE: SAS initialization used:
      real time           1.10 seconds
      cpu time            0.90 seconds
```

Figure 2.1 The SAS window environment

The following windows open by default:

- Enhanced editor window,
- Log window,
- Explorer window,
- Results window and
- Output window.

SAS programs (see Section 2.2.2) are created, by typing in program or batch code in the **enhanced editor window.** More than one enhanced editor window may open at the same time. The program code is submitted in order to execute the SAS program. This is done by clicking on the submit button or by selecting the *Run* → *Submit* option from the pull-down menus. The whole program or a portion of it may be submitted. The mouse is used to select a block of program code. If the submit button is clicked, then only the selected program code is submitted. If no selection is made then all the program code is submitted and the whole program is executed. A SAS program is saved as a SAS file with a *(*.sas)* extension by selecting the *File → Save As* option from the pull-down menus and then specifying the appropriate folder and file name. A **SAS file** is defined in general as a file that is created and used in the SAS window environment.

A record of the current SAS session is kept in the **log window**. The information in this window includes the following:

- the program code of SAS programs that were recently submitted,
- information about the SAS files that were recently read or created,
- the execution information and results of the SAS programs and
- the relevant error and warning messages about submitted SAS programs.

The different types of messages in the log window are printed in different colours. Program code is printed in black, successful or confirmation messages in blue, warning messages in green and error messages in red. This feature makes it easy to check the successful execution of a SAS program. The log window is activated by any of the following methods: Click with the mouse anywhere in the log window, or on the *Log* button, or select the *View → Log* option from the pull-down menus. The log window plays a vital role in the implementation of a successful SAS program.

The **explorer window** is used to view the SAS library structure (see Section 2.2.3) of the SAS window environment. The contents of a SAS library are called SAS catalogs. A **SAS catalog** is defined as a SAS file that can be stored in a SAS library. The contents of a Windows folder for example the *My Computer* folder are also viewed in the explorer window. This is a new feature of SAS Version 9. The explorer window is activated by either clicking with the mouse on the *Explorer* button or by selecting the *View → Explorer* option from the pull-down menus.

When SAS programs are executed, various results are created. The names of these results are listed in the **results window.** Similar to the explorer window it is activated by either clicking with the mouse on the *Results* button or by selecting the *View* → *Results* option from the pull-down menus.

The results that were created by SAS programs are viewed in the **output window**. The output window is activated by either clicking with the mouse on the *Output* button or by selecting the *View* → *Output* option from the pull-down menus.

Each of the enhanced editor, output and log windows is cleared by selecting the *Edit* → *Clear All* option from the pull-down menus or by selecting the *Clear All* 🗋 icon.

2.2 SAS structures

SAS structures are created and used within the SAS window environment. The basic structures, namely SAS data sets, SAS programs and SAS libraries are discussed in detail in this section.

2.2.1 SAS data sets

In order to use a set of data in the SAS window environment for any type of analysis it has to be stored in a special type of SAS file called a **SAS data set**. SAS programs are used to create these data sets. Each SAS data set is divided into two portions, namely the **data portion** and the **descriptor portion**.

The data portion contains the data values of the data set, in the form of a rectangular table. Each column refers to a variable and each row contains one record, with one observation for each variable. The data values are either character or numeric values. A variable has a character data type if its data values contain any combination of letters, numbers or special characters. If the data values of a variable contain only numbers with a decimal point and minus sign optional, it is of a numeric data type.

The descriptor portion of the SAS data set contains **general information** about the data set, as well as, **variable specific information**. The general information consists of the name of the SAS data set, the date and time of its creation, the number of variables it contains and the number of observations (rows) of all the variables. The variable specific information contains the name, the label, the position, the length and the data type of each variable in the data set.

Every observation of every single variable in the SAS data set must be a **valid data value**. Missing character values are left blank and missing numeric values are replaced by a dot. The name of a SAS data set has a maximum length of 32 characters and has to start with a letter or an underscore "_". The rest of the name may consist of any combination of characters, numbers and underscores.

2.2.2 SAS programs

A **SAS program** is a block of **program code** that is saved in a SAS file with a *(*.sas)* extension. SAS programs are used to create SAS data sets and to perform many different types of analyses.

The program code is entered in the enhanced editor window and consists of a sequence of **program steps**. A program step is a sequence of one or more **program statements**.

The only two kinds of program steps are:

- Proc steps and
- Data steps

The **Proc step** activates a pre-written SAS program called a **SAS procedure**. This step is used to perform many different kinds of tasks, some examples are: *Proc Print*, *Proc Contents, Proc Import, Proc Report, Proc Compile* and *Proc Risk*. For example, the *Proc Print* procedure displays the data portion of a SAS data set in the output window, whilst the *Proc Contents* procedure displays the descriptor portion.

The **Data step** is used to create new SAS data sets, modify existing SAS data sets and to transform or import raw data files into SAS data sets.

A Data step starts with a *Data* statement, a Proc step with a *Proc* statement and both ends with a *Run* statement. Each statement is followed by a semicolon. The general form of these two steps is illustrated in Figure 2.2.

Components of the **Data Step**	Components of the **Proc Step**
Data ______________________;	**Proc** ______________________;
______________________;	______________________;
....	....
______________________;	______________________;
Run;	**Run**;

Figure 2.2: General form of the Proc and Data steps

Commentary statements are used to provide insight into some program statements. In the first method, the symbols **/*** and ***/** opens and closes the commentary statement respectively. The comments are typed between these symbols. The star symbol, “ ***** ”, may also be used. The program code between the star and the next semi-colon is viewed as commentary code. Program code is also not case sensitive and program statements may be used in the same line or it may extent over different lines. The statements may also begin or end in any column of the enhanced editor window.

2.2.3 SAS libraries

SAS files are created and used in the SAS window environment and are stored physically on the hard drive. These files are grouped in folders according to certain considerations. SAS files are not grouped according to folders in the SAS window environment. They are grouped to a structure analogous to folders called **SAS libraries**. A SAS library is defined as a collection of SAS files that are grouped as a unit in the SAS window environment. In the creation of a SAS library a certain folder is specified. All the SAS files in the folder are then contents of the SAS library as well.

Temporary and **permanent** SAS libraries exist. The user has access to a temporary SAS library called *Work* at the start of each SAS session. Suppose that SAS files are created and grouped in this library during the current SAS session, then at the end of the session these SAS files are automatically erased. The user also has access to two permanent SAS libraries with names *Sasuser* and *Sashelp* at the start of each SAS session. SAS files that are stored in these SAS libraries are available in subsequent SAS sessions. Program code may be used to create additional permanent libraries. Suppose further that SAS files are created and grouped in one of the user-defined libraries, then these SAS files are physically stored on the hard drive in the folder that was specified during the creation of the additional library. At the end of the current session the SAS files are still kept in the folder on the hard drive, but are not grouped as a unit (SAS library) by the SAS Window Environment anymore. It is thus necessary to create the SAS library (not the SAS files) again at the start of the next SAS session.

The ***Libname*** statement is used to create an additional permanent library. The statement is a global statement outside of the Data step or Proc step and has the following general form:

```
Libname Libref   "Name-and-location-of-folder" ;
```

The name of the SAS library is specified in *Libref* and has a maximum length of eight characters. The name and path of the folder in which the SAS files are stored is specified in the *"Name-and-location-of-folder"* option. As mentioned previously, the name of the SAS library specified in *Libref* remains assigned to the SAS files in the folder only for the current SAS session.

In Section 2.1 a **SAS catalog** has been described as a SAS file that can be stored in a SAS library. Whenever it is necessary to specify a certain SAS catalog in program code, it is referred to by its name and the SAS library it is grouped in. The dot symbol, ".", is used to separate the *Libref* from the name of the SAS catalog. The reference takes on the following form:

Libref.Name-of-Catalog

A wide variety of information is stored as entries in a SAS catalog. SAS data sets and Risk Dimensions environments are examples. Each entry in a SAS catalog is called an object. A SAS data set is only one file and the SAS catalog has only one object. The Risk Dimensions environment usually consists of many files. Thus the corresponding SAS catalog has many objects. Risk Dimensions environments are discussed in more detail in Chapter 4.

2.3 An example

Example 2.1: The working of the SAS window environment:

This example illustrates the use of SAS structures like SAS data sets, SAS libraries and SAS programs within the SAS window environment. The use of the Data step, Proc step and the *Libname* statement within a SAS program is also illustrated.

The objectives of the example are to:

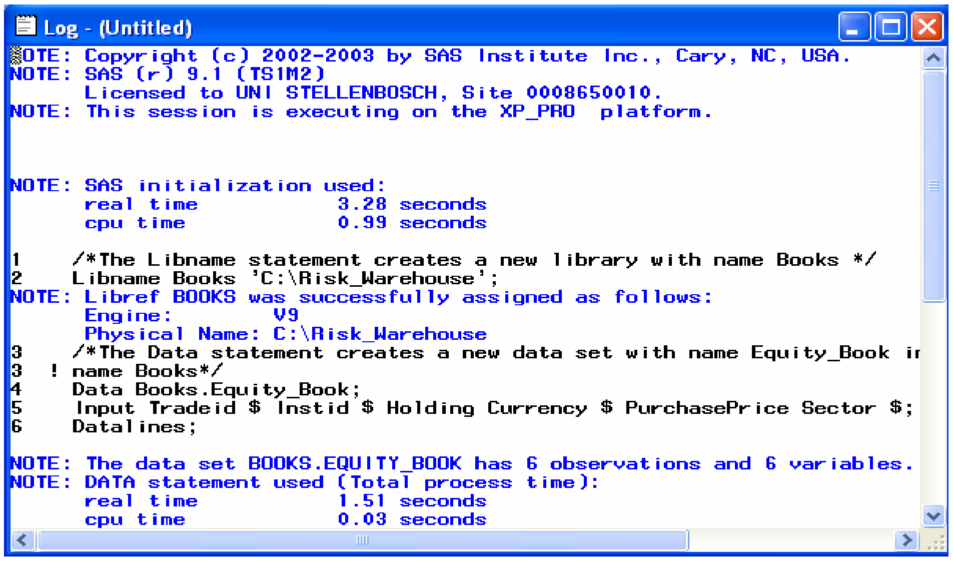
- Create a SAS library, named *Books*.
- Create a new SAS data set, with name *Equity_Book* that contains information about the open positions held on equities. The names of the variables in *Equity_Book* are *Tradeid, Instid, Holding, Currency, PurchasePrice* and *Sector*.
- Illustrate the data portion and descriptor portion of the *Equity_Book* data set, using the SAS procedures *Proc Contents and Proc Print*.

The *Input* statement is used in the Data step to declare the variables in the SAS data set. The name of the variables are listed in this statement and the variables that are of character data type are specified with a dollar "*$*" symbol. The *Datalines* statement precedes the data input in the program code and initiates the creation of a new SAS data set. The SAS data set *Equity_Book* is referred to by its two-level name *Books.Equity_Book* in the program code. Program Code 2.1 is typed in the enhanced editor window.

Program Code 2.1: The Working of the SAS window environment:

```
/*The Libname statement creates a new library with name Books. The
location and name of the folder that is assigned to the library is
C:\Risk_Warehouse */
Libname Books "C:\Risk_Warehouse";
/*The Data statement creates a new data set, named Equity_Book in the
SAS library with name Books*/
Data Books.Equity_Book;
Input Tradeid $ Instid $ Holding Currency $ PurchasePrice Sector $;
Datalines;
2003_001 ASA 10131 ZAR  3050 FINANCIALS
2003_002 ASA  7030 ZAR  2860 FINANCIALS
2003_003 SOL  1032 ZAR 10010 RESOURCES
2003_004 SOL  2938 ZAR 10394 RESOURCES
2003_005 SOL  3920 ZAR  9843 RESOURCES
2003_006 ASA  1022 ZAR  3560 FINANCIALS
Run;
/*The data portion of Equity_Book is viewed in the output window*/
Proc Print Data= Books.Equity_Book;
Run;
/*The descriptor portion of Equity_Book is viewed in the output window*/
Proc Contents Data= Books.Equity_Book;
Run;
```

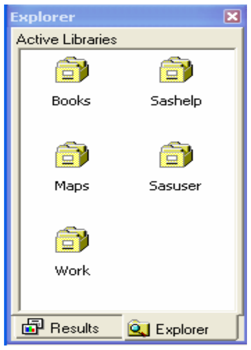
The submit button is used to submit the program code. The contents of the log window are checked for confirmation of a successful execution of the SAS program. Figure 2.3 contains part of the output in the log window.



```
NOTE: Copyright (c) 2002-2003 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) 9.1 (TS1M2)
      Licensed to UNI STELLENBOSCH, Site 0008650010.
NOTE: This session is executing on the XP_PRO  platform.



NOTE: SAS initialization used:
      real time           3.28 seconds
      cpu time            0.99 seconds

1     /*The Libname statement creates a new library with name Books */
2     Libname Books 'C:\Risk_Warehouse';
NOTE: Libref BOOKS was successfully assigned as follows:
      Engine:        V9
      Physical Name: C:\Risk_Warehouse
3     /*The Data statement creates a new data set with name Equity_Book ir
3   ! name Books*/
4     Data Books.Equity_Book;
5      Input Tradeid $ Instid $ Holding Currency $ PurchasePrice Sector $;
6     Datalines;

NOTE: The data set BOOKS.EQUITY_BOOK has 6 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           1.51 seconds
      cpu time            0.03 seconds
```

Figure 2.3: The Log Window

The log window contains no warning or error messages and a successful execution of the SAS program is confirmed. The explorer window is used to view the changes that have been made to the SAS library structure. The explorer window is activated and the *Libraries* icon is clicked on. The new library with name *Books* is viewed in Figure 2.4. The contents of this library is viewed by clicking on the icon with name *Books.* The SAS catalog or SAS data set *Equity_Book* is contained in this library as illustrated in Figure 2.5.



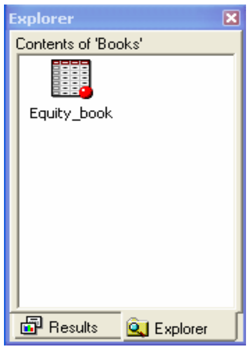
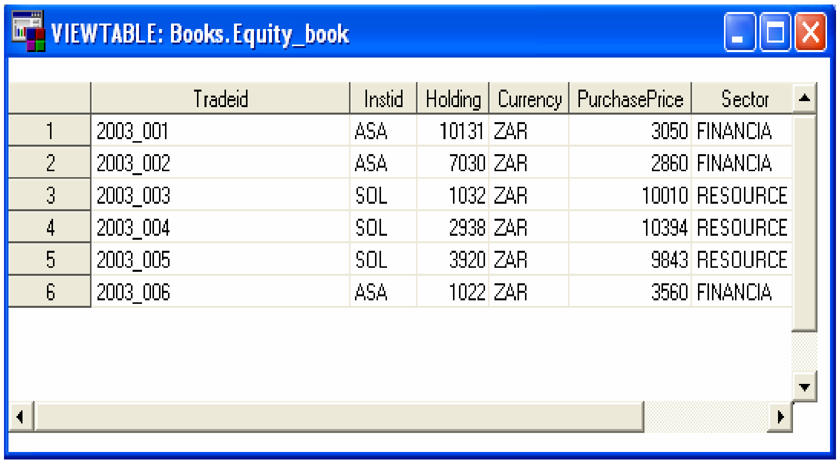


Figure 2.4: The explorer window   Figure 2.5 The explorer window

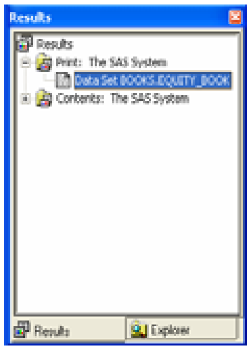
The data portion of the SAS data set *Equity_Book* is viewed in Figure 2.6. The illustration is created by clicking on the SAS data set icon with name *Equity_Book*.



	Tradeid	Instid	Holding	Currency	PurchasePrice	Sector
1	2003_001	ASA	10131	ZAR	3050	FINANCIA
2	2003_002	ASA	7030	ZAR	2860	FINANCIA
3	2003_003	SOL	1032	ZAR	10010	RESOURCE
4	2003_004	SOL	2938	ZAR	10394	RESOURCE
5	2003_005	SOL	3920	ZAR	9843	RESOURCE
6	2003_006	ASA	1022	ZAR	3560	FINANCIA

Figure 2.6: The SAS data set *Equity_Book*

The names of the results that were created by the execution of *Proc Print* and *Proc Contents* are viewed in the results window, as illustrated in Figure 2.7. A specific result is selected by clicking on the *Print: The SAS System* option and then double clicking on the data set option that has subsequently opened. The possible results names that may be selected are viewed in Figure 2.8.



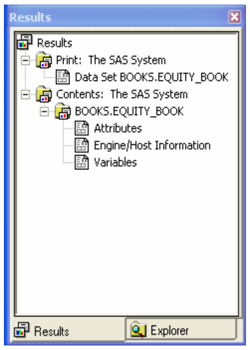
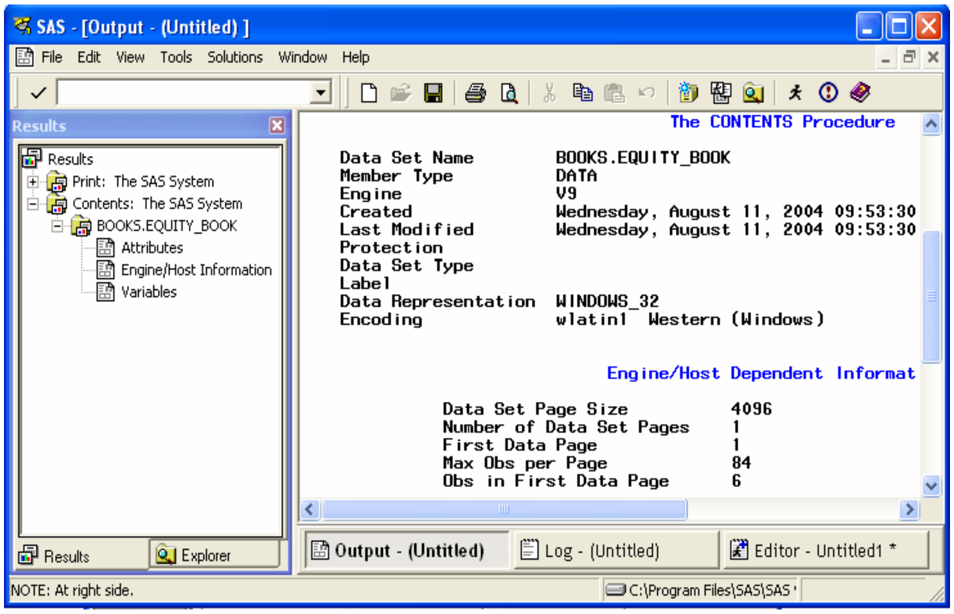


Figure 2.7: The results window   Figure 2.8 The results window

The results that were created by *Proc Contents* are viewed in the output window by applying the following steps: Double click on the *Contents: The SAS System* option in the results window and double click again on the *Data Set Books.Equity_Book* option that becomes available. The results are divided into three parts, *Attributes, Engine/Host Information* and *Variables.* Each separate result is viewed in the output window by clicking on the appropriate option in the results window. The options in results window and the corresponding output in the output window are viewed in Figure 2.9.

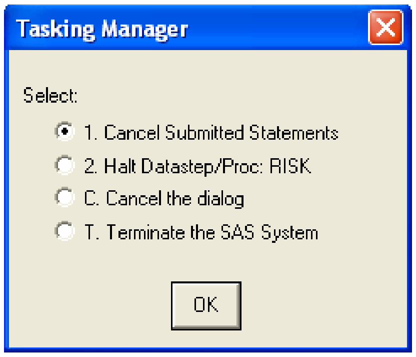


```
                                    The CONTENTS Procedure

Data Set Name          BOOKS.EQUITY_BOOK
Member Type            DATA
Engine                 V9
Created                Wednesday, August 11, 2004 09:53:30
Last Modified          Wednesday, August 11, 2004 09:53:30
Protection
Data Set Type
Label
Data Representation    WINDOWS_32
Encoding               wlatin1  Western (Windows)


                                 Engine/Host Dependent Informat

              Data Set Page Size          4096
              Number of Data Set Pages    1
              First Data Page             1
              Max Obs per Page            84
              Obs in First Data Page      6
```

Figure 2.9: Results in the Output window

The full set of results in the output window is viewed by activating the output window and scrolling down from the top to the bottom.

If a block of program code is unsuccessfully submitted, the SAS program may enter an endless loop. The program can be halted by either pressing the *Control* and *Break* button on the keyboard simultaneously or by clicking on the icon. The *Tasking Manager* window as illustrated in Figure 2.10 opens.


Tasking Manager

Select:

[✓ 1. Cancel Submitted Statements]
2. Halt Datastep/Proc: RISK
C. Cancel the dialog
T. Terminate the SAS System

Figure 2.10: The Tasking Manager window

The selection of either the *Cancel Submitted Statements* option or the *Halt Datastep/Proc: Risk* option cancels the endless loop. It is necessary to click on the *OK* button to confirm the decision.

2.4 Summary

The use of the SAS window environment and the creation of basic SAS structures in it are an integral part of implementing a successful risk management system. The knowledge gained from this chapter is used in all the subsequent chapters.

3

CASE STUDY DEFINITION AND WORKSPACE PREPARATION STEPS

A fictitious **case study** is used to illustrate the working of SAS Risk Dimensions. The case study is described in detail in Section 3.1 and is used in the subsequent chapters to explain and illustrate the Risk Dimensions concepts that are discussed.

The rest of the chapter focuses on the creation of a **workspace** that is necessary before Risk Dimensions is opened. This entails:

- the creation of raw data files,
- the creation of a physical workspace on the hard drive,
- the creation of SAS libraries and
- the conversion of raw data files into SAS data sets.

3.1 Case study definition

Consider a fictitious South African company, named *Activegrowth Limited*. It is an investment company that actively trades in the following financial instruments: options, futures, government bonds, interest rate swaps and equities. The company has recently acquired SAS Risk Dimensions and plans to use the software package to calculate various risk measures or analyses on a daily basis.

Options and futures form the derivatives portfolio. The equity portfolio is divided into the resources, financials and industrial sectors. The resources sector consists primarily of mining companies, whilst banks and life insurers form the financial sector. Manufacturing companies such as the steel giant, Iscor forms part of the industrial sector. The government bonds and interest rate swaps, form the interest rate derivatives portfolio. The investments in each of the financial instruments may also be viewed as sub-portfolios.

All the open positions that are held in financial instruments are recorded in trade books at the end of every trading day. Three trade books are used and for each trade book a corresponding raw data file is created. One trade book is used for equities, futures and options, whilst interest rate swaps and government bonds are each recorded separately. Real life equities, government bonds and futures are used in the case study. The interest rate swaps and options that are used are, however, fictitious. The historical closing prices of the relevant market variables, for example share prices, interest rates and swap rates are also saved in raw data files. All the raw data files that were mentioned above contain information that is used in the valuation of financial instruments and execution of various risk analyses. These raw data files are discussed in more detail in Section 3.2.1.

Risk management systems in Risk Dimensions may be designed to calculate both market and credit risk measures. Only market risk measures are calculated for *Activegrowth Limited*. The measures include Value at Risk (VaR), sensitivity analysis, scenario analysis, profit/loss curves and profit/loss surfaces. These analyses are performed on the whole portfolio or selected portions. Monte Carlo simulation and historical simulation are used in the different Value at Risk calculation methodologies.

The mark-to-market value (MtM) of the portfolio and the results obtained by calculating the risk measures are included in batch reports. The reports supply information about the whole portfolio, as well as, the certain specified sub-portfolios. Five sub-portfolios are used in the case study reports. Each sub-portfolio consists of all the position held in the same financial instrument. An example is the futures portfolio. The reports are further easy to interpret and are used in risk management decisions. The calculation of risk measures and the creation of reports are discussed in Chapters 10 and 11 respectively.

3.2 The preparation of the workspace

The case study and any other similar business problems require certain preparation steps before Risk Dimensions is opened. Some of these steps are performed outside the SAS window environment, for example in Microsoft Windows or Microsoft Excel. Other steps are performed in the SAS window environment.

The following four steps provide a guide to the process of preparing the appropriate workspace:

1. The creation of raw data files.
2. The creation of a physical workspace on the hard drive.
3. The creation of the appropriate SAS libraries.
4. The conversion of raw data files into SAS data sets.

Each of these steps is discussed in detail below. Also, see the graphical illustration of these steps in A1 of the Appendix.

3.2.1 The creation of raw data files

A **raw data file** is defined as a data file that is created outside of the SAS window environment. Observed data such as trade book information or market information is usually captured in raw data files. Raw data files are converted into SAS data sets. The data values contained in the SAS data sets are used during the execution of risk analyses like Value at Risk (VaR). The captured data usually has to be adjusted or transformed into the correct form before an analysis can be executed. The steps that are necessary to prepare the data may be done inside the SAS window environment using SAS programs or outside SAS in a software program like Microsoft Excel. A combination of both software packages is usually used.

The raw data files that are necessary for the case study are subsequently discussed. These files are updated at the end of each trading day. Raw data files are created for the trade books of the company and for the relevant market information. The creation of each raw data file is discussed separately.

A raw data file that contains information about the open positions held in equities, futures and options is created. The raw data file, named ***Tradebooksource*** typically consists of variables (column names) like *Insttype* (instrument type)*,* *Instid* (instrument identification*), Holding, Currency, Premium (price paid for instrument), OptType* (type of option)*, Strike, Enddate, Contractprice, Sector, Book*, *Shortposition, Shareprice* and *Underlying*. Each row in the data set contains information about one position taken in a financial instrument. Most of the columns and observations of the raw data file are presented in the following extract:

Insttype	**Instid**	**Short**	**Holding**	**Premium**	**Sector**	**Strike**	**Enddate**	**Opttype**	**Cprice.**
Equity	SOL_001	0	400	94.7	Res	.	.	.	.
Equity	SLM_002	1	8500	7.8	Fin	.	.	.	.
Equity	ASA_001	0	2800	30	Fin	.	.	.	.
Equity	ASA_002	0	2000	28.5	Fin	.	.	.	.
Equity	OML_001	0	8200	12.5	Fin	.	.	.	.
Equity	OML_002	1	1400	10.1	Fin	.	.	.	.
Future	ASA_QM4	0	10000	.	.	.	17-Jun-04	.	46.59
Future	OML_Q43	0	15000	.	.	.	17-Jun-04	.	11.93
Future	SLM_Q42	0	4000	.	.	.	17-Jun-04	.	9.54
Future	SOL_Q41	1	14000	.	.	.	17-Jun-04	.	97.86
Future	SOL_Q42	0	12000	.	.	.	17-Jun-04	.	103.29

Extract from *Tradebooksource* continues…

Insttype	**Instid**	**Short**	**Holding**	**Premium**	**Sector**	**Strike**	**Enddate**	**Opttype**	**Cprice.**
Option	ASA_O02	0	6000	4.67	.	40	29-Jun-04	EC	.
Option	ASA_O06	0	10000	3.8	.	33	14-Sep-04	EP	.
Option	SOL_O04	1	5000	5	.	88	18-Oct-04	EC	.
Option	SOL_O05	0	6000	4.3	.	93	27-Jul-04	EP	.
Option	SLM_O05	0	18000	1.2	.	8.8	15-Aug-04	EC	.

The raw data file by name ***Bondbook*** contains records of open positions held in government bonds. The variables or column names are *InstType*, *Instid, Shortposition, Notional, Holding, Maturiydate, Currency, Coupfreq* (coupon frequency)*, Coupon, Red_Amount* (Redemption amount) and *Premium.* The four observations and most of the columns of the raw data file are included in the following extract:

InstType	**Instid**	**Notional**	**Holding**	**MaturityDate**	**Coupon**	**Premium**	**Red_Amount**
Gov_Bond	R153_1	100	100	8/31/2010	0.13	85	100
Gov_Bond	R153_2	100	600	8/31/2010	0.13	84.3	100
Gov_Bond	R133_1	100	2400	9/15/2007	0.15	70	100
Gov_Bond	R177_1	100	2500	5/15/2007	0.095	98	100

Another raw data file is created to capture information about the open positions held in interest rate swaps. The name of the raw data file is ***Swapbook*** and consists of the variables or column names *InstType*, *Instid*, *Shortposition*, *Notional*, *Holding*, *MaturityDate*, *Fromdate*, *Rcvetype* (type of payment received), *FixRate*, *Ftr_name* (name of floating rate), *Currency* and *Coupfreq* (Frequency of payment exchanges). The three observations and most of the columns of the raw data file is included in the following extract:

InstType	**Instid**	**Short**	**Notional**	**MaturityDate**	**Fromdate**	**Rcvetype**	**FixRate**	**Ftr_name**
Int_Swap	DB_IS_01	0	150000	12/17/2006	12/17/2003	Floating	0.06	JB_6_MTH
Int_Swap	IB_IS_02	0	1000000	4/17/2007	4/17/2004	Fixed	0.1	JB_6_MTH
Int_Swap	IB_IS_03	0	850000	5/17/2005	11/17/2003	Floating	0.065	JB_6_MTH

The data values of the variables in the above mentioned three raw data files are used in the valuation and the grouping of the financial instruments in the portfolio.

Historical time series data of market variables such as equity prices, interest rates and volatility in equity prices, are contained in the raw data file, ***Market_History***. Each row contains the observations of all the market variables for one day. The record in the last row (observation) contains the current date and data values that are used to calculate the current value (mark-to-market value) of the instruments in the portfolio. Three observations of some of the variables or columns in the raw data file are viewed in the following extract:

Date	ASA	AGL	SLM	SOL	Vol_ASA	Vol_AGL	Vol_SLM	Vol_SOL	JB_6_MTH
05/11/2004	44.9	135.45	8.73	102.5	0.209979	0.297494	0.219129	0.286326	0.08291
05/12/2004	44.5	133.5	8.41	97.5	0.21226	0.297322	0.218985	0.287471	0.08297
05/13/2004	45	133.5	8.55	99	0.210693	0.296292	0.224054	0.28974	0.08303

Another raw data file, namely, ***Logreturns*** is created. Each historical equity price in *Market_History* is divided by the closing value of the equity on the previous day. The logarithm of the ratio, is then calculated for all the equities on each available date. An extract of the raw data file follows:

Date	Ret_ASA	Ret_AGL	Ret_ISC	Ret_OML	Ret_SLM	Ret_Sol
05/11/2004	0.023663	-0.00111	-0.0047	0.002601	-0.00229	0.030208
05/12/2004	-0.00895	-0.0145	0.004386	-0.01659	-0.03734	-0.05001
05/13/2004	0.011173	0	0.028049	0.012249	0.01651	0.015267

Large financial institutions are the market makers in the interest rate swap market. The institutions are prepared to quote for a number of different maturities a bid and an offer for the fixed rate they will exchange for a floating rate. The bid is the fixed rate which the market makers will pay in exchange for a floating rate, whilst the offer is the fixed rate which they have to receive in exchange for a floating rate. The average of the bid and offer rates is called the **swap rate**. Swap rates are observed in the market for varying maturities. The swap rates, together with the **JIBAR** (Johannesburg Interbank Agreed Rate) rates are used to construct a yield curve. The yield curve is used in the valuation of financial instruments like options, futures and government bonds. The constituents of the yield curve are called **zero rates**. A zero rate is the risk-free rate of interest that can be earned for a specified maturity. The JIBAR rates are transformed into zero rates for maturities, ranging from one month to one year. The swap rates are transformed into zero rates using the so-called bootstrap method (cf. Hull (2003)). The maturities of these zero rates range from one to ten years. The length of the interval between subsequent maturities is six months. The zero rates are calculated for each historical date in the *Market_History* data set and are stored in the raw data file with name ***Yieldcurve_data***. The calculation of zero rates is carried out in Microsoft Excel. An example of a variable (column name) in this data set is *Unstd_3_Year*. This column contains the zero rates for historical dates corresponding to a maturity value of 3 years. It is important to note that use of JIBAR rates and swap rates in the yield curve construction leads to zero rates that are not strictly risk-free rates of interest. This is true as swap rates contain a risk premium, because the counter party may default on the interest rate payments. Two observations of some of the variables in the raw data file are viewed in the next extract:

UNSTD_1_MTH	UNSTD_3_YEAR	UNSTD_5_YEAR	UNSTD_8_YEAR	UNSTD_10_YEAR
0.080231191	0.080628	0.088802	0.094315	0.095034
0.08011199	0.082553	0.090761	0.095316	0.095722

A raw data file named ***Scenariodata*** that will be used in scenario simulation (see Section 10.2.7) is created. The column names are, with the exception of a few the same as the *Market_History* raw data file. The values in the raw data file are user-defined and are not observed in any financial market. Two of the observations of some of the variables are viewed in the following extract:

ASA	AGL	ISC	SOL	Vol_ASA	Vol_AGL	Vol_ISC	Vol_SOL	JB_6_MTH
45	133.1	38	115	0.18	0.31	0.356	0.31	0.08
35.3	140	32.8	99	0.18	0.25	0.323	0.313	0.0655

The raw data files may be stored as different types of files. Only comma delimited files with the (**.csv*) extension are discussed. This type of file is easily created in Microsoft Excel. The raw data files that are created are

Tradebooksource.csv**, **Swapbook.csv**, **Bondbook.csv**, **Yieldcurve_data.csv**, **Logreturns.csv**, **Scenariodata.csv and ***Market_History.csv***. These data files have to be updated on a daily basis, before new risk analyses are executed.

3.2.2 The creation of a physical workspace on the hard drive

A physical workspace in the form of a folder with subfolders is created on the hard drive of the computer. A folder with a name, for example ***Risk_Warehouse,*** is created at a suitable space on the hard drive. Within this folder sub-folders with the following names are created:

- *Rawfiles,*
- *Riskdata,*
- *Env,*
- *Source,*
- *Local,*
- *Output and*
- *Models.*

The raw data files that are created in Section 3.2.1 are stored in the *Rawfiles* folder. For the case study *Tradebooksource.csv, Swapbook.csv, Bondbook.csv, Logreturns.csv, Yieldcurve_data.csv, Scenariodata.csv* and *Market_history.csv* are stored in this folder. All the SAS data sets that are created in the SAS window environment are stored in the *Riskdata* folder. Risk environments that are created are stored in the *Env* folder. SAS programs are stored in the *Source* folder. The *Local* folder is used during the data-driven variable registration process, discussed in Section 6.5. Statistical models are fitted on the historical data values of some of the market variables in Chapter 8. The models are used to predict future values of the market variables. The output from these fitted models is stored in the *Models* folder. The execution of risk analyses creates output data sets. These data sets contain a variety of information regarding the risk analyses that are used in the risk management system. The output data sets are stored in the *Output* folder. Windows Explorer is used to create the folders mentioned above. The role that folders play in the workspace is graphically illustrated in A1 in the Appendix.

3.2.3 The creation of the appropriate SAS libraries

The necessary folders in the workspace were created in the previous section. Corresponding SAS libraries are created for some of these folders in this section. A SAS library is necessary if a SAS catalog for example a SAS data set is stored in a folder by the execution of program code. If the SAS file or other type of file is stored in the folder by another method, no SAS library is necessary. Thus, the way in which files are stored in a folder determines the need for a corresponding SAS library.

SAS programs and raw data files are saved directly in the *Source* and *Rawfiles* folders respectively. SAS programs are saved by using the *File → Save* option from the pull-down menus in the SAS window environment. Raw data files are created outside the SAS window environment and are stored directly in the *Rawfiles* folder. Thus, there is no need to create corresponding SAS libraries for the *Source* and *Rawfiles* folders.

SAS programs are used to create SAS catalogs, for example SAS data sets, in the SAS window environment. These files have to be stored on the hard drive. SAS programs cannot store these files directly in folders, but are able to create and group these files in SAS libraries. It is necessary to assign a SAS library to a corresponding folder on the hard drive. This enables the SAS catalogs to be stored in the appropriate folders via SAS libraries during the execution phase of the SAS program. In the case study example SAS data sets and other SAS catalogs are stored in the *Riskdata*, *Env, Models, Output* and *Local* folders.

Thus, it is necessary to create corresponding SAS libraries with names *Riskdata*, *Env, Models, Output* and *Local.*

User-defined macro variables are used to shorten the program code that is necessary to create and use SAS libraries. A global program statement namely, ***%Let*** is used to assign macro variables and has the following general form:

```
%Let   macro-variable-name  string-assigned-to-macro-variable;
```

For example, the name *RiskPath* is given to the location of the physical workspace *C:\Risk_Warehouse* in the following code*:*

```
%Let RiskPath = C:\Risk_Warehouse;
```

Whenever it is necessary to use the character string *C:\Risk_Warehouse* in program code, only the macro variable name, with a preceding ampersand “&”, i.e. *&Riskpath* is used.

With the code:

```
%Let Rawfiles = &Riskpath\Rawfiles;
```

a new macro variable with name *Rawfiles* is created and refers to the character string *C:\Risk_Warehouse\Rawfiles* that specifies the location of the folder on the hard drive.

The following user-defined macro variables are also assigned:

```
%Let Source = &RiskPath\Source;
%Let RiskData = &RiskPath\RiskData;
%Let Env = &RiskPath\Env;
%Let Local = &RiskPath\Local;
%Let Models = &RiskPath\Models;
%Let Output = &RiskPath\Output;
```

The following ***Libname*** statements use the user-defined macro variables to create SAS libraries with names *RiskData*, *Local, Env, Models* and *Output* in the SAS window environment:

```
Libname RiskData "&RiskData";
Libname Local "&Local";
Libname Env "&Env";
Libname Models "&Models";
Libname Output "&Output";
```

3.2.4 The conversion of raw data files into SAS data sets

3.2.4.1 Overview

Raw data files are updated at the end of each trading day. The data files typically contain market, position and other information that are used in the execution of risk analyses. In order to use information contained in the data files within the SAS window environment or in Risk Dimensions, the information has to be contained in SAS data sets. Thus, it is necessary to create algorithms that are able to convert raw data files into SAS data sets on a daily basis.

Raw data files may be stored as different types of files. The column structure determines the type of raw data file. Each column refers to a variable and each row contains a record of one observation of each column. Only raw data files where the data values are separated by commas (comma separated value files) are discussed in this document.

Two conversion methods are considered. The first method is to use the **Data step** in program code. It requires a large amount of program code, but has the advantage that the format of the data values in the raw data files can easily be changed. The second method, is to use the **Import Wizard**. It is a point and click graphical interface that creates the desired SAS data sets quite easily. It also produces program code that may be used to repeat the process for similar raw data files. Although this method is suitable for various different types of raw data files, it is not very adaptive to the format in which data values are stored in the raw data file. The data values have to be in a pre-specified format, without any missing values in the first row. If this criterion is not met, this method either fails to create a SAS data set or it creates a corrupt data set.

In the case study, seven files, namely ***Tradebooksource.csv**, **Swapbook.csv**, **Bondbook.csv**, **Logreturns.csv**, **Yieldcurve_data.csv**, **Scenariodata.csv*** and ***Market_History.csv*** that are stored in the *Rawfiles* folder are converted or imported into SAS data sets. The names of the corresponding SAS data sets are ***Tradebook**, **Swapbook**, **BondBook**, **Logreturns**, **Yieldcurve_data**, **Scenariodata*** and ***Market_History*** and are stored in the SAS library named *RiskData.*

3.2.4.2 Conversion with the Data step

The Data step may be used to convert or import raw data files into SAS data sets. The basic structure of the Data step necessary to read a raw data file is illustrated in Figure 3.1.

Components of Data Step	
Data	______________________;
Infile	______________________;
Input	______________________;
Informat	______________________;
Format	______________________;
	....
	______________________;
Run;	

Figure 3.1: The basic structure of Data step used in the conversion process

The components (statements) of the data step will subsequently be discussed:

The ***Data*** statement names, the SAS data set that is being created, together with the SAS library that it is grouped in. The general form of the *Data* statement follows:

```
Data Libref.Catalog;
```

The name of the SAS library is specified in the *Libref* option and the name of the SAS data set in the *Catalog* option. The SAS data set is stored in the temporary library *Work* if the *Libref* option is omitted.

The ***Infile*** statement specifies the path and the name of the raw data file that is converted. The ***Input*** statement specifies the names of the variables (columns) in the raw data file that are included in the SAS data set. The ***Informat*** statement and ***Format*** statement are optional statements. The Data step has a default way of reading data from the raw data file. A SAS instruction, namely a **SAS informat** is used to manually define the way the data values are read from the raw data file. SAS informats may be included in the *Input* statement or in the *Informat* statements. Another SAS instruction, namely a **SAS format** is used to alter the way in which the data values of SAS data sets are presented. SAS formats are included in the optional *Format* statements. More than one *Informat* statement and *Format statement* may be included in the Data step.

SAS informats

It is necessary to study SAS informats in detail, before the implementation of this instruction is illustrated in the Data step. SAS informats are only used within ***Informat*** statements in this section.

The *Informat* statement has the following general form:

```
Informat  Name-of-variable  SAS-informat-used;
```

The name of the variable that is read from the raw data file is specified in the *Name-of-variable* option. The *Informat* statement makes use of the *SAS* informat specified in *SAS-informat-used* option to read the variable in a manually defined way. An *Informat* statement may be specified for each variable that is read.
A SAS informat has the following general form:

$informat-name.d

The "*$*" symbol is only used when the variable that is imported is of character data type. The decimal point "*.*" is a required delimiter and the "*d*" is an optional argument. The "*d*" is used to indicate the number of decimal places that are allocated for a numeric variable.

SAS informats are used to read numeric values, character values, numeric values with dollars or commas as well as date values. A SAS informat specifies the type of data to be read, as well as, the maximum length of the data values of the variable. For example, if a data value in the raw data file is a name with nine characters and the SAS informat is specified with a maximum length of eight, then only the first eight characters are read and stored in the SAS data set.

The following *Informat* statement reads the values of the variable named *Holding*. The SAS informat, *8.0*, specifies that the data values contained in *Holding* are numeric, of maximum length 8 and that no allowance is made for any decimal points.

```
Informat Holding 8.0;
```

The values of the variable *Currency* are read with the following *Informat* statement:

```
Informat Currency $3.;
```

The SAS informat, "*$3.*", specifies that the data values in *Currency* are read as character values with a maximum length of 3 characters.

Date values are stored as special numeric values within the SAS environment. A date value is interpreted as the number of days between the specified date and the first of January 1960. Suppose the data values in the variable *Enddate* is stored in the *DDMMYY* format, for example *13MAY04*, in the raw data file, then the data values of this variable are read with the following *Informat* statement:

```
Informat Enddate date7.;
```

The data values in the *Enddate* variable are stored in the *DDMMYY* format. The SAS informat namely *date7.* reads these values from the raw data file and ensures that it is stored as valid SAS date values, for example 16204.

Table 3.1 contains a list of SAS informats that are frequently used in either the *Informat* or *Input* statements. The column named, *Raw Data Value,* illustrates the format in which the data values are stored in the raw data file. The SAS informats that are used to manipulate the way in which the data values are read are viewed in the *Informat* column. *The SAS Data Value* column illustrates the format in which the data values are stored in the SAS data set.

Table 3.1: SAS informats

Raw Data Value											**Informat**		**SAS Data Value**							
			1	2	3	4	5	6	7	→	**8.0**	→		1	2	3	4	5	6	7
		1	2	3	4	.	5	6	7	→	**8.0**	→	1	2	3	4	.	5	6	7
			1	2	3	4	5	6	7	→	**8.2**	→	1	2	3	4	5	.	6	7
		1	2	3	4	.	5	6	7	→	**8.2**	→	1	2	3	4	.	5	6	7
					B	O	B	B	Y	→	**$8.**	→	B	O	B	B	Y			
					B	O	B	B	Y	→	**$Char8.**	→				B	O	B	B	Y
			$	1	2	,	3	4	5	→	**Comma7.0**	→				1	2	3	4	5
0	5	/	1	3	/	2	0	0	4	→	**MMDDYY10.**	→				1	6	2	0	4
		0	5	/	1	3	/	0	4	→	**MMDDYY8.**	→				1	6	2	0	4
	1	3	M	A	Y	2	0	0	4	→	**Date9.**	→				1	6	2	0	4
1	3	/	0	5	/	2	0	0	4	→	**DDMMYY10.**	→				1	6	2	0	4

SAS formats

A SAS instruction, called a **SAS format** is used to alter the format in which the data values of the newly created SAS data set are displayed. The SAS formats are included in the ***Format*** statement within the Data step.

The general form of the format statement is:

Format Name-of-variable SAS-format-used;

The name of the variable that is viewed is specified in the *Name-of-variable* option. The *Format* statement uses the SAS format, specified in *SAS-format-used* to view the data values of the variable in the SAS data set in a certain format.

SAS formats have the following general form:

Format-name.d

The decimal point "." is a required delimiter and the "*d*" argument is optional. The *"d"* argument indicates the number of decimal places that are used in the presentation of the data values of a numeric variable.

Consider the variable named *Holding* again. The data values contained in this variable have been saved in the SAS data set as numeric data values with maximum length of eight and no decimal points, for example 12345678. If the data set is viewed, the data values are presented well, if commas are added, for example 12,345,678. This is achieved by using the following *Format* statement:

```
Format Holding comma10.;
```

If the format width is not specified large enough in the SAS format, the displayed value is automatically adjusted to fit into the width. If the format statement was:

```
Format Holding comma8.;
```

the displayed value would have been 12345678, since the commas could not fit into the maximum length specified.

When the SAS data set is opened the value of the variable *Enddate* can be viewed in any other data format. The following format statement lets the data values of the *Enddate* variable be viewed as *DDMMYYYY*, for example *13MAY2004*.

```
Format Enddate date9.;
```

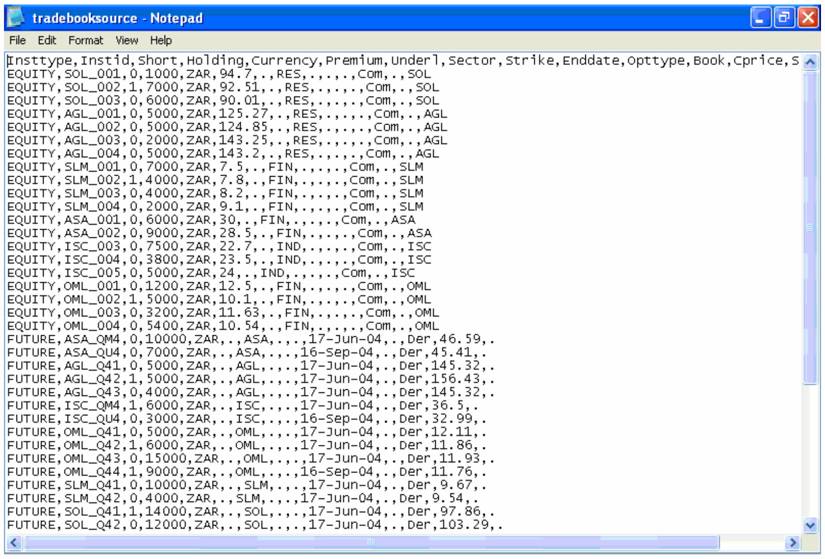
Table 3.2 contains a list of SAS formats that are used in *Format* statements. The *Stored Value* column illustrates the format in which the data values are stored in a SAS data set. The SAS formats that are used in *Format* statements are contained in the *Format* column. The format in which the data values in the SAS data set are presented is viewed in the *Displayed Value* column.

Table 3.2: SAS formats

Stored Value		**Format**		**Displayed Value**
12345.6789	→	Comma12.2	→	12,345.68
12345.6789	→	12.2	→	12345.68
12345.6789	→	Dollar12.2	→	$12,345.68
12345.6789	→	Dollar8.2	→	12345.68
12345.6789	→	Dollar5.2	→	12345
16024	→	MMDDYY6.	→	051304
16024	→	MMDDYY10.	→	05/13/2004
16024	→	Date7.	→	13MAY04
16024	→	Date9.	→	13MAY2004
16024	→	Worddate.	→	May 13, 2004
16024	→	Weekdate.	→	Thursday, May 13, 2004

The conversion of *Tradebooksource.csv* into a SAS data set

Consider the raw data file, by name *Tradebooksource* that has been created as a comma separated value file with a (**.csv*) extension. The data values in each row (observation) are separated by a comma. *Notepad* is used to view the structure of the *T*radebooksource.csv file and this is illustrated in Figure 3.2.



```
Insttype,Instid,Short,Holding,Currency,Premium,Underl,Sector,Strike,Enddate,Opttype,Book,Cprice,S
EQUITY,SOL_001,0,1000,ZAR,94.7,.,RES,.,.,.,.,Com,.,SOL
EQUITY,SOL_002,1,7000,ZAR,92.51,.,RES,.,.,.,.,Com,.,SOL
EQUITY,SOL_003,0,6000,ZAR,90.01,.,RES,.,.,.,.,Com,.,SOL
EQUITY,AGL_001,0,5000,ZAR,125.27,.,RES,.,.,.,.,Com,.,AGL
EQUITY,AGL_002,0,5000,ZAR,124.85,.,RES,.,.,.,.,Com,.,AGL
EQUITY,AGL_003,0,2000,ZAR,143.25,.,RES,.,.,.,.,Com,.,AGL
EQUITY,AGL_004,0,5000,ZAR,143.2,.,RES,.,.,.,.,Com,.,AGL
EQUITY,SLM_001,0,7000,ZAR,7.5,.,FIN,.,.,.,.,Com,.,SLM
EQUITY,SLM_002,1,4000,ZAR,7.8,.,FIN,.,.,.,.,Com,.,SLM
EQUITY,SLM_003,0,4000,ZAR,8.2,.,FIN,.,.,.,.,Com,.,SLM
EQUITY,SLM_004,0,2000,ZAR,9.1,.,FIN,.,.,.,.,Com,.,SLM
EQUITY,ASA_001,0,6000,ZAR,30,.,FIN,.,.,.,.,Com,.,ASA
EQUITY,ASA_002,0,9000,ZAR,28.5,.,FIN,.,.,.,.,Com,.,ASA
EQUITY,ISC_003,0,7500,ZAR,22.7,.,IND,.,.,.,.,Com,.,ISC
EQUITY,ISC_004,0,3800,ZAR,23.5,.,IND,.,.,.,.,Com,.,ISC
EQUITY,ISC_005,0,5000,ZAR,24,.,IND,.,.,.,.,Com,.,ISC
EQUITY,OML_001,0,1200,ZAR,12.5,.,FIN,.,.,.,.,Com,.,OML
EQUITY,OML_002,1,5000,ZAR,10.1,.,FIN,.,.,.,.,Com,.,OML
EQUITY,OML_003,0,3200,ZAR,11.63,.,FIN,.,.,.,.,Com,.,OML
EQUITY,OML_004,0,5400,ZAR,10.54,.,FIN,.,.,.,.,Com,.,OML
FUTURE,ASA_QM4,0,10000,ZAR,.,ASA,.,.,.,17-Jun-04,.,Der,46.59,.
FUTURE,ASA_QU4,0,7000,ZAR,.,ASA,.,.,.,16-Sep-04,.,Der,45.41,.
FUTURE,AGL_Q41,0,5000,ZAR,.,AGL,.,.,.,17-Jun-04,.,Der,145.32,.
FUTURE,AGL_Q42,1,5000,ZAR,.,AGL,.,.,.,17-Jun-04,.,Der,156.43,.
FUTURE,AGL_Q43,0,4000,ZAR,.,AGL,.,.,.,17-Jun-04,.,Der,145.32,.
FUTURE,ISC_QM4,1,6000,ZAR,.,ISC,.,.,.,17-Jun-04,.,Der,36.5,.
FUTURE,ISC_QU4,0,3000,ZAR,.,ISC,.,.,.,16-Sep-04,.,Der,32.99,.
FUTURE,OML_Q41,0,5000,ZAR,.,OML,.,.,.,17-Jun-04,.,Der,12.11,.
FUTURE,OML_Q42,1,6000,ZAR,.,OML,.,.,.,17-Jun-04,.,Der,11.86,.
FUTURE,OML_Q43,0,15000,ZAR,.,OML,.,.,.,17-Jun-04,.,Der,11.93,.
FUTURE,OML_Q44,1,9000,ZAR,.,OML,.,.,.,16-Sep-04,.,Der,11.76,.
FUTURE,SLM_Q41,0,10000,ZAR,.,SLM,.,.,.,17-Jun-04,.,Der,9.67,.
FUTURE,SLM_Q42,0,4000,ZAR,.,SLM,.,.,.,17-Jun-04,.,Der,9.54,.
FUTURE,SOL_Q41,1,14000,ZAR,.,SOL,.,.,.,17-Jun-04,.,Der,97.86,.
FUTURE,SOL_Q42,0,12000,ZAR,.,SOL,.,.,.,17-Jun-04,.,Der,103.29,.
```

Figure 3.2: The raw data file, Tradebooksource.csv

The data step in Program Code 3.1 is used to convert the raw data file, *Tradebooksource.csv*, into a SAS data set with the name *Tradebook* that is stored in the SAS library with name *Riskdata*.

Program Code 3.1: The conversion of *Tradebooksource.csv*

```
/*The SAS data set named Tradebook is created in the SAS library
Riskdata*/
Data Riskdata.Tradebook;
/*The raw data file is specified as Tradebooksource.csv and is imported
from the C:\Risk_Warehouse\Rawfiles folder*/
Infile "&Rawfiles\Tradebooksource.csv"
/*The delimiter is set to a comma and the Firstobs option specifies that
the first row of data values is row number 2*/
Delimiter = ',' Firstobs = 2;
```

Program Code 3.1 continues…

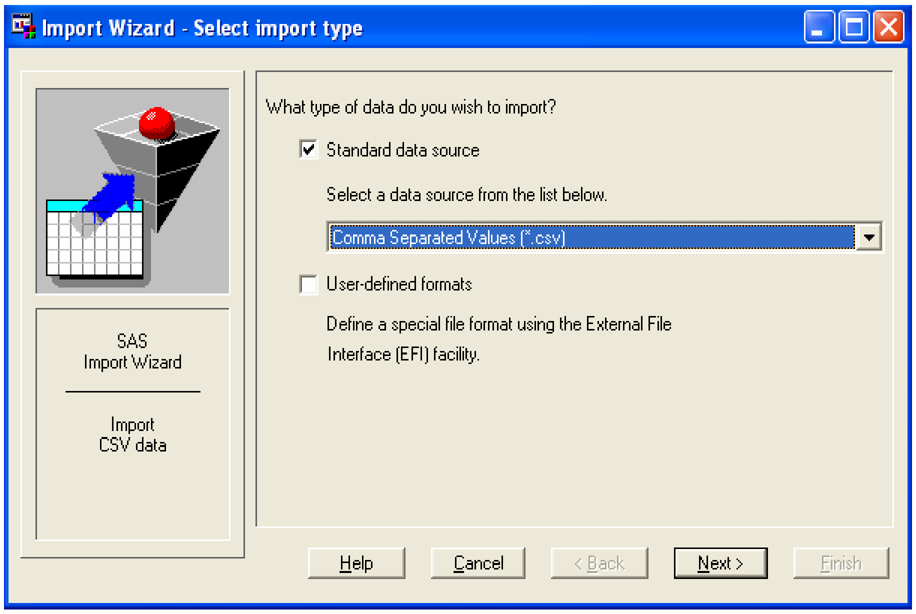
```
/*The SAS informats contained in the Informat statements manually
define the way in which the data values in the raw data file are read*/
Informat Insttype $6.;
Informat Instid $9. ;
Informat Shortposition 1.;
Informat Holding 5.;
Informat Currency $3.;
Informat Premium 8.;
Informat Underlying $4.;
Informat Sector $10.;
Informat Strike 5.;
Informat Enddate date7.;
Informat Opttype $3.;
Informat Book $9.;
Informat ContractPrice 6.;
Informat SharePrice $10.;
/*The SAS format that is included in the Format statement specifies the
format in which the data values of Enddate is presented*/
Format Enddate date9.;
/*The names of all the variables that are included in the SAS data set
are specified in the Input statement. The names of variables with a
character data type are followed with a "$" sign.*/
Input  Insttype       $
       Instid         $
       Shortposition
       Holding
       Currency       $
       Premium
       Underlying     $
       Sector         $
       Strike
       Enddate
       Opttype        $
       Book           $
       Contractprice
       Shareprice $;
Run;
```

It is important to note that in Program Code 3.1 the SAS informats were used in the *Informat* statements.

3.2.4.3 Import Wizard

The **import wizard** is a tool that is used to convert or import raw data files into SAS data sets. The mouse is used to point and click on options in a series of windows. The tool has the capability of converting different types of raw data files into SAS data sets. Examples are comma separated value files (**.csv*), Microsoft Excel spreadsheets (**.xls*) and delimited files (**.**). The import wizard can also generate program code that is used to perform subsequent conversions of the same type. To ensure that a correct SAS data set is created, all the date values in the raw data file are stored in the following format: *MM/DD/YYYY*.

The steps that are necessary to convert the raw data file, named *Market_History.csv* into a SAS data set are discussed in the next few pages. The first step is selecting the *File* → *Import Data* option from the pull-down menus. The *Select import type* window as illustrated in Figure 3.3, opens.


Import Wizard - Select import type

What type of data do you wish to import?

☑ Standard data source

Select a data source from the list below.

Comma Separated Values (*.csv)

☐ User-defined formats

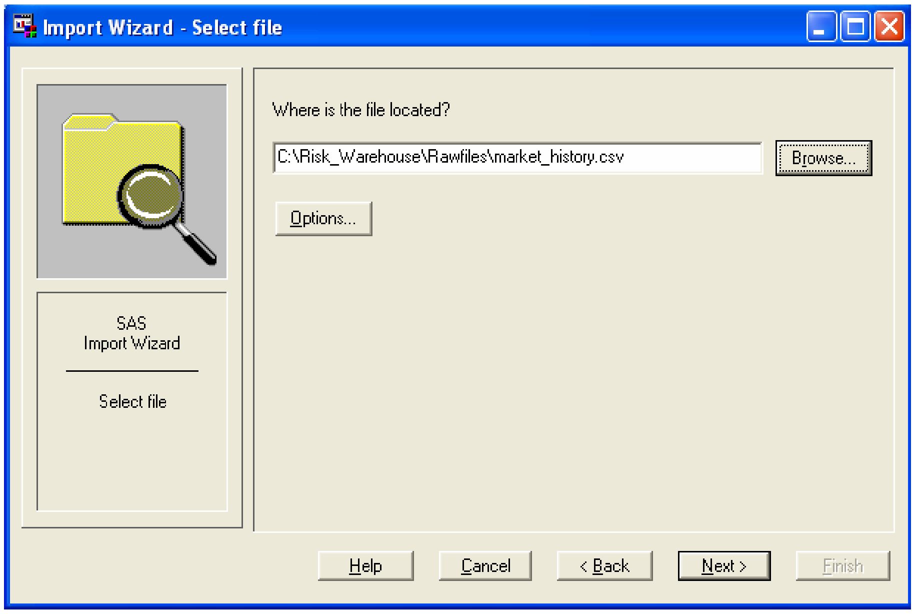
Define a special file format using the External File Interface (EFI) facility.

SAS Import Wizard

Import CSV data

Figure 3.3: The *Select Import Type* window

The selection of *Comma Separated Values* is made in the *Data Source List* option. After clicking on the *Next* button, the *Select file* window opens as illustrated in Figure 3.4.

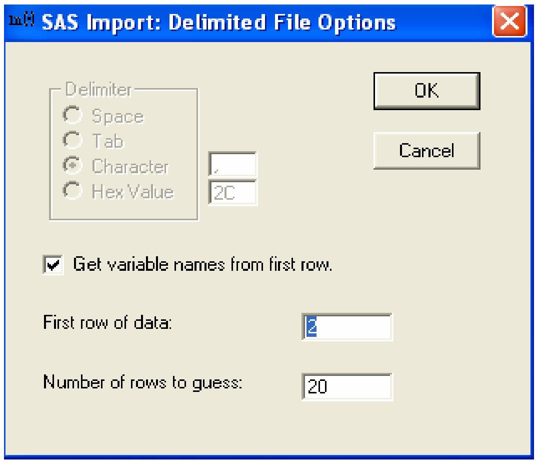


Where is the file located?

C:\Risk_Warehouse\Rawfiles\market_history.csv

Figure 3.4: The *Select File* window

The physical location of the file to be imported or converted is specified and the *Next* button is clicked. Hence, the *Options* button is selected and the *Delimited File Options* window as illustrated in Figure 3.5 opens.



Delimiter
- Space
- Tab
- [✓ Character] ,
- Hex Value 2C

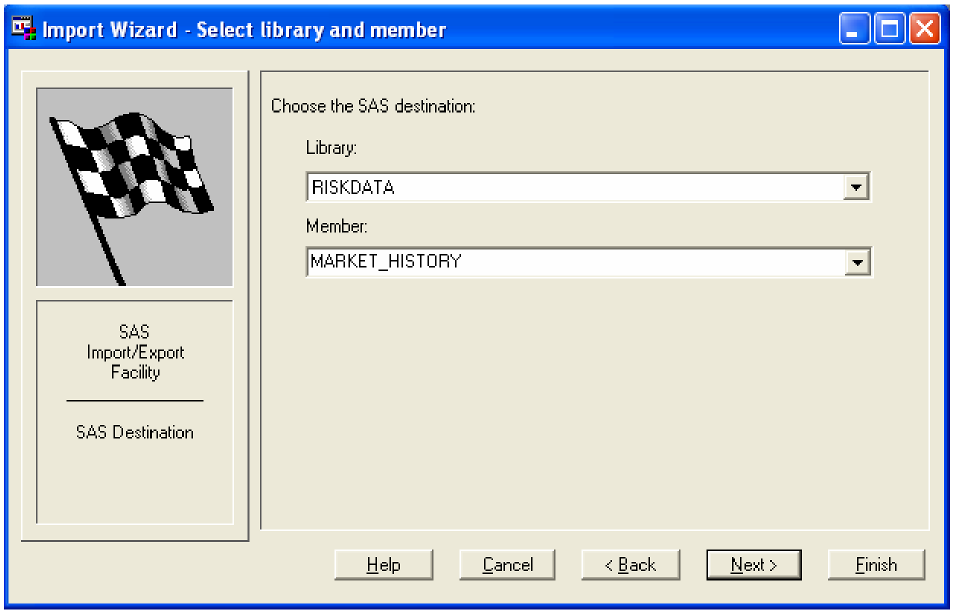
[✓] Get variable names from first row.

First row of data: 2

Number of rows to guess: 20

Figure 3.5: The *Delimited File Options* window

The value for the *First row of data* option is set to “2” and the *OK* button is clicked. The *Select File* window (Figure 3.4) opens again and the *Next* button is clicked. The *Select Library and member* window opens as illustrated in Figure 3.6.



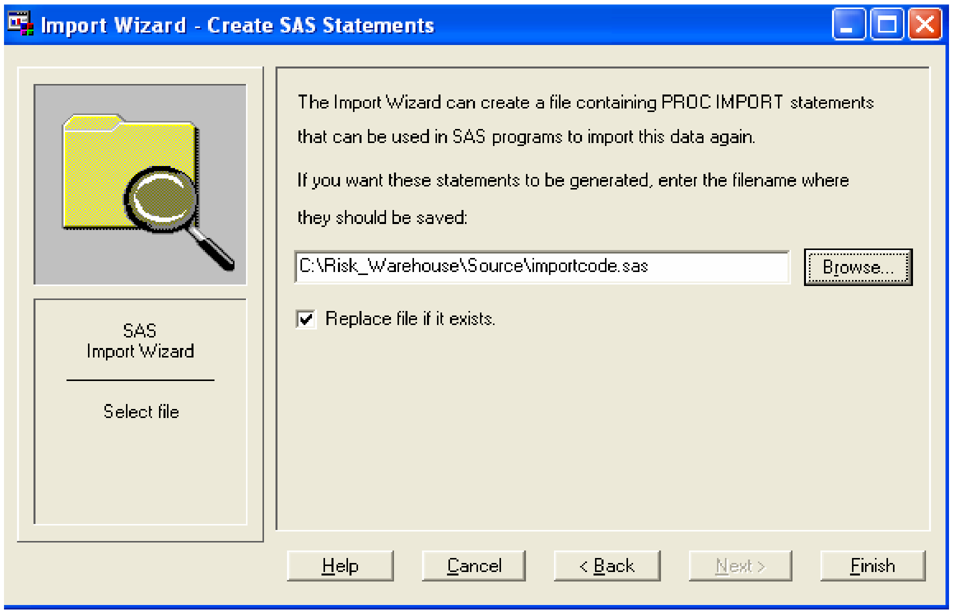
Choose the SAS destination:

Library: RISKDATA

Member: MARKET_HISTORY

Figure 3.6: The *Select Library and member* window

The destination library is chosen as *Riskdata* and *Market_History* is specified as the name of the new SAS data set. The *Next* button is clicked once again and the *Create SAS statements* window as illustrated in Figure 3.7 opens.



The Import Wizard can create a file containing PROC IMPORT statements that can be used in SAS programs to import this data again.

If you want these statements to be generated, enter the filename where they should be saved:

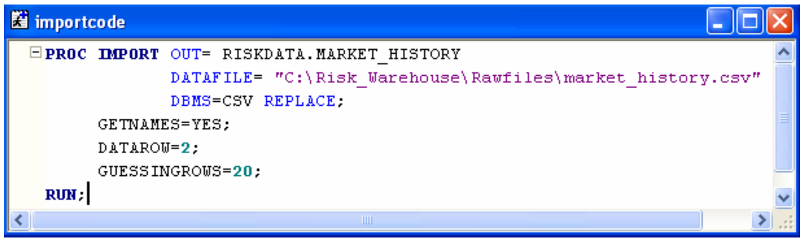
C:\Risk_Warehouse\Source\importcode.sas

[✓] Replace file if it exists.

Figure 3.7: The *Create SAS statements* window

The program code that is created, is saved in the *Source* folder in a SAS file named *Importcode.sas*. The *Finish* button is clicked to finish the conversion process. The log window is browsed to check for any error or warning messages, as well as, to confirm that the raw data file has been successfully converted into a SAS data set.

The program file, named *Importcode.sas* is stored in the *C:\Risk_Warehouse\Source* folder and is illustrated in Figure 3.8.



```
PROC IMPORT OUT= RISKDATA.MARKET_HISTORY
            DATAFILE= "C:\Risk_Warehouse\Rawfiles\market_history.csv"
            DBMS=CSV REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     GUESSINGROWS=20;
RUN;
```

Figure 3.8: The SAS program file *Importcode.sas*

The SAS procedure *Proc Import* executes the importation of the data set. The name of the new SAS data set is specified in the *Out* option. The *Datafile* option is used to specify the physical location of the raw data file that will be converted. The type of raw data file is specified in the *DBMS* option. The *CSV* specification in this option means that a comma separated value file with an (*.csv*) extension is imported. The *Getnames* and *Datarow* statements refer to the headings and starting point of actual data values in the raw data file.

The procedure *Proc Import* in Program Code 3.2 is used to convert the remaining raw data files in the case study into the SAS data sets with the same names. The newly created SAS data sets are grouped in the SAS library, named *Riskdata*. The statements and options in the procedure *Proc Import* are used in the same way as in Figure 3.8.

Program Code 3.2: The conversion of raw data files into a SAS data sets

```
Proc Import Out = Riskdata.Market_History
            Datafile = "&Rawfiles\Market_history.csv"
            DBMS = CSV Replace;
            Getnames = yes;
            Datarow = 2;
Run;
Proc Import Out= Riskdata.Swapbook
            Datafile = "&Rawfiles\Swapbook.csv"
            DBMS = CSV Replace;
     Getnames = yes;
     Datarow = 2;
Run;
Proc Import Out = Riskdata.Bondbook
            Datafile = "&Rawfiles\Bondbook.csv"
            DBMS = CSV Replace;
     Getnames = yes;
     Datarow = 2;
Run;
Proc Import Out = Riskdata.Yieldcurve_data
            Datafile = "&Rawfiles\Yieldcurve_data.csv"
            DBMS = CSV Replace;
     Getnames = yes;
     Datarow = 2;
Run;
Proc Import Out = Riskdata.Logreturns
            Datafile = "&Rawfiles\Logreturns.csv"
            DBMS = CSV Replace;
     Getnames = yes;
     Datarow = 2;
Run;
Proc Import Out = Riskdata.Scenariodata
            Datafile = "&Rawfiles\Scenariodata.csv"
            DBMS = CSV Replace;
            Getnames = yes;
            Datarow = 2;
Run;
```

It is simpler and quicker to use *Proc Import* on a daily basis than to use the import wizard again.

3.3 Summary

The case study that is used in the remainder of the document was outlined in broad terms in Section 3.1. The first steps in implementing a successful risk management system for the case study were illustrated in the rest of this chapter. Appropriate raw data files, folders and SAS libraries were created. The methods that are available to convert or import a raw data file into a SAS data set were discussed in detail in Section 3.2.4. Seven raw data files were also converted into SAS data sets. A graphical illustration of the workspace is available in A1 of the Appendix.

The next step in the process, namely the creation of risk environments is discussed in the next chapter.

4

RISK ENVIRONMENTS

4.1 Introduction

Consider the case study as described in Section 3.1. The company *ActiveGrowth* invested in a portfolio of financial instruments and has the need to calculate various risk measures that portray the company's exposure to risk. Various structures are created in Risk Dimensions in order to implement a suitable risk management system. Examples of the creation of Risk Dimensions structures include:

- the registering of variables containing market and position data values,
- the creation of pricing functions and instrument types,
- the registration of market and portfolio data sources,
- the creation of risk factor models and
- the creation of risk analysis structures.

Each risk management problem is unique and requires a different combination of Risk Dimensions structures. All the structures that are created to address a particular part of a risk management problem are grouped together in a special Risk Dimensions structure called a **risk environment**. One or more risk environments may be used in a particular risk management system.

The main part of SAS Risk Dimensions is the **graphical user interface** or the **GUI**. It is used to view the existing structures in a particular risk environment. The GUI may also be used to create new risk environments and additional structures in existing risk environments. A number of pull-down menus that are specifically designed to assist the creation of new structures exists. The structures are created one at a time and the occasional text input is also required.

The GUI of each risk environment is divided into six different tree views. Each tree has a separate tab and is used to display a certain group of Risk Dimensions structures. The structures are listed in the following tree views:

- The *Analysis* tree,
- the *Portfolios* tree,
- the *Market Data* tree,
- the *Risk Models* tree,
- the *Report Gallery* tree and
- the *Configuration* tree.

SAS programs are also used to create structures in risk environments. The program code in the SAS procedure ***Proc Risk*** is used to create Risk Dimensions structures.

The process of creating new Risk Dimensions structures in the GUI is easy to understand and execute. The method of using program code in *Proc Risk* is more difficult at first. This method is, however, very effective if the amount of structures that needs to be created is quite large.

The optimal solution is to use mainly program code, but in certain areas where it is easier, the GUI method should be used. The GUI is used to view the structures that were created by *Proc Risk* in risk environments.

It is possible to create more than one risk environment for a risk management problem. Risk environments are able to inherit information from one another. This is useful, for example, when pricing functions created in one environment are used in other environments. The pricing functions are stored at one physical location on the hard drive, but are used in both environments. A risk environment inheritance structure may also be specified. This is determined by the order in which the environments are inherited. This topic is not covered in detail, as the case study requires only one risk environment.

4.2 The creation of a new environment in the GUI

The following steps illustrate the creation of a new risk environment in Risk Dimensions. The GUI is used to create the new environment. If the SAS window environment is open, the icon that activates the GUI is found in the top left-hand corner of the screen:

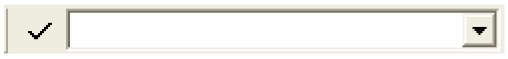


Figure 4.1: The GUI Icon

The word *risk* is typed into the icon and the ✓ button is clicked.

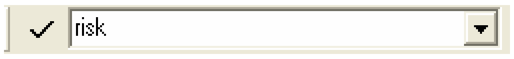
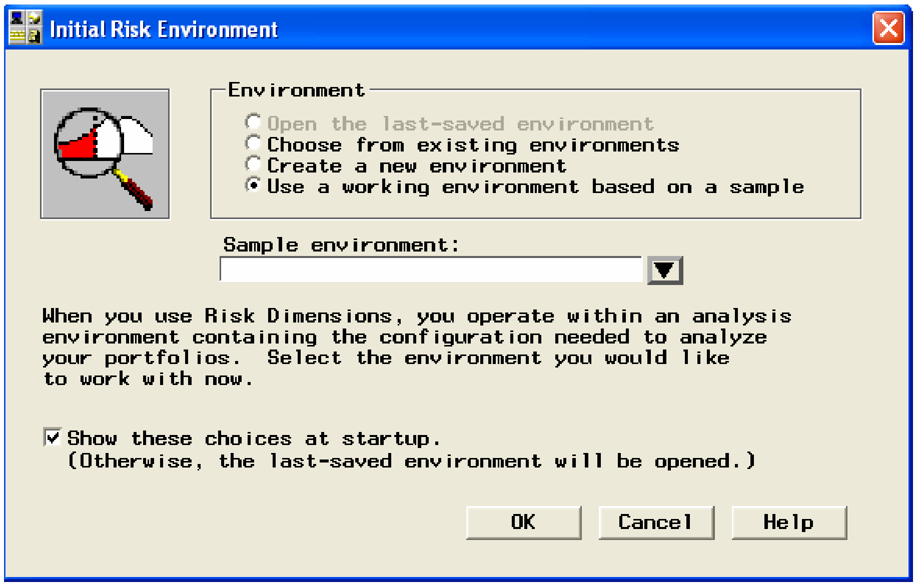


Figure 4.2: The GUI Icon with the word *risk*

The *Initial Risk Environment* window opens, as illustrated in Figure 4.3.


Initial Risk Environment

Environment
- Open the last-saved environment
- Choose from existing environments
- Create a new environment
- [✓ Use a working environment based on a sample]

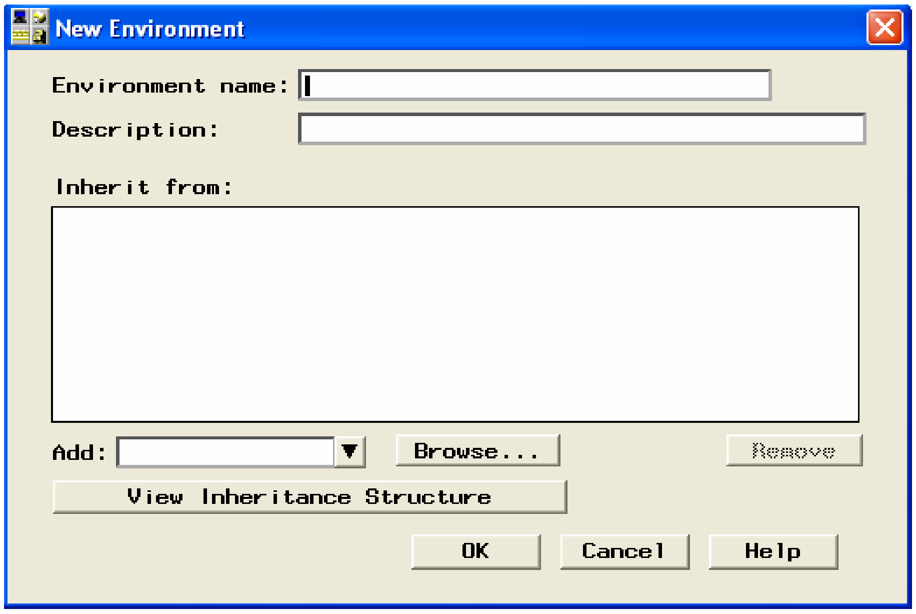
Sample environment:

When you use Risk Dimensions, you operate within an analysis environment containing the configuration needed to analyze your portfolios. Select the environment you would like to work with now.

[✓] Show these choices at startup.
(Otherwise, the last-saved environment will be opened.)

Figure 4.3: The *Initial Risk Environment* window

The *Create a new environment* option is selected in this window and the *OK* button is clicked. The *New Environment* window opens:


New Environment

Environment name:

Description:

Inherit from:

Add:

Figure 4.4: The *New Environment* window

The environment name *Casestudy_Env* and the description *The Case Study Environment* is specified in this window. No inheritance structure is specified and the *OK* button is clicked. The *Casestudy_Env* risk environment opens in the *Analysis* tree as illustrated in Figure 4.5. Specific types of Risk Dimensions structures may be viewed later under the *Specifications Library*, *Analysis Projects* and *Results* options. In a similar way the other Risk Dimensions structures are grouped under different options in the other trees. The contents of the other trees are viewed by clicking on the name of a tab, for example *Portfolios*.

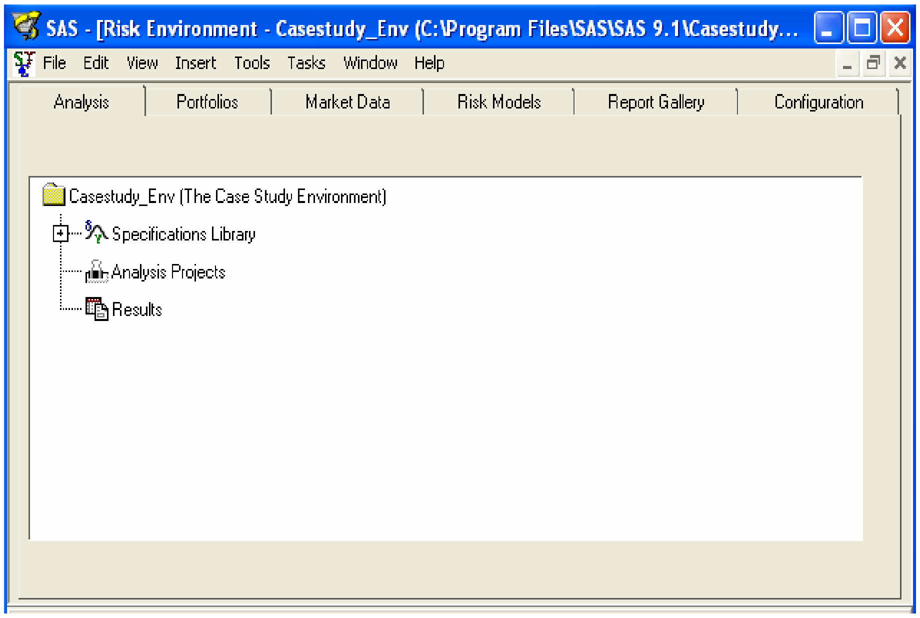
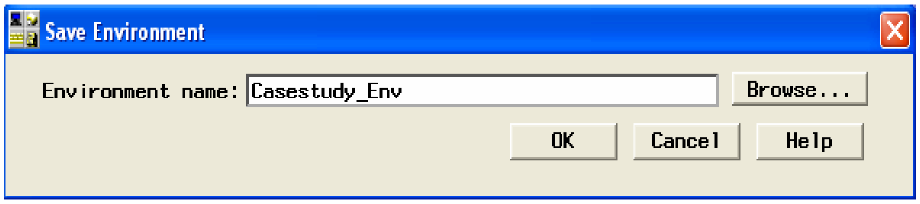

SAS - [Risk Environment - Casestudy_Env (C:\Program Files\SAS\SAS 9.1\Casestudy...
File Edit View Insert Tools Tasks Window Help
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration
Casestudy_Env (The Case Study Environment)
Specifications Library
Analysis Projects
Results

Figure 4.5: The *Analysis* tree of *Casestudy_Env*

The *Casestudy_Env* is saved in an unspecified default folder. In order to save the risk environment in the *Env* folder that was created in Section 3.2.2, the *File → Save Environment as* option from the pull-down menu, is used.

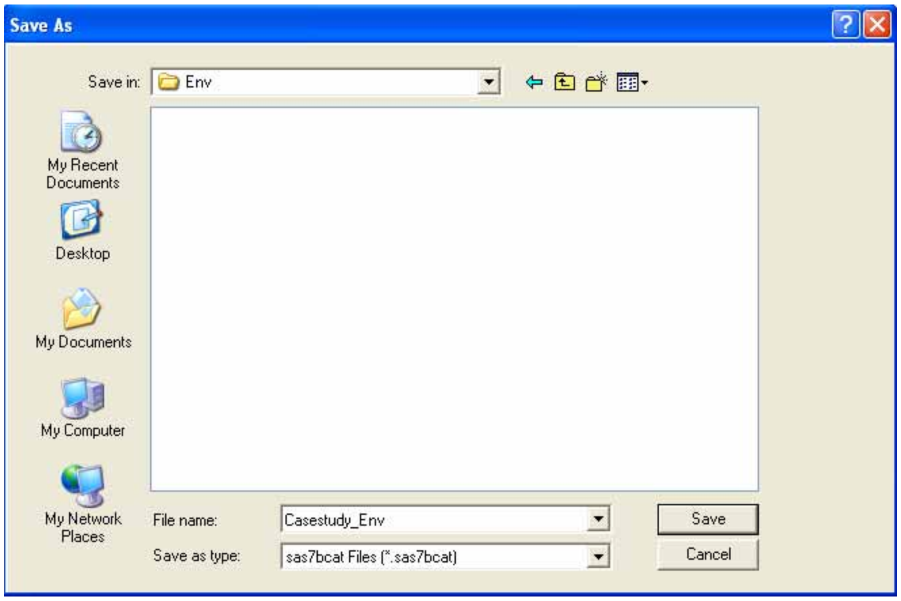
The *Save Environment* window opens in Figure 4.6.


Save Environment

Environment name: Casestudy_Env

Figure 4.6: The *Save Environment* window

The *Browse* button is clicked on and the *Save As* window as illustrated in Figure 4.7 opens:


Save As

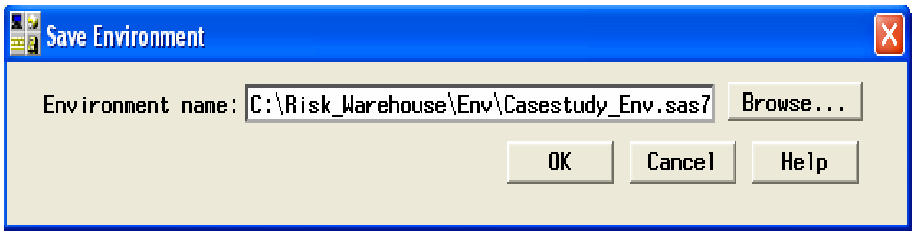
Save in: Env

File name: Casestudy_Env

Save as type: sas7bcat Files (*.sas7bcat)

Figure 4.7: The *Save As* window

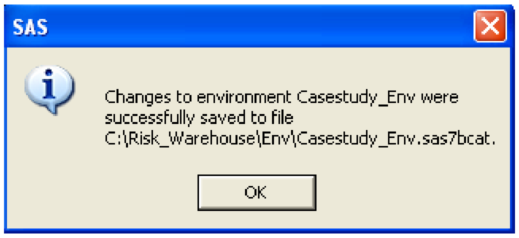
The *Env* folder is specified in the *Save in* option. The file name is specified as *Casestudy_Env* with an (**.sas7bcat*) extension. The *Save* button is clicked. The *Save Environment* window opens, as illustrated in Figure 4.8.



C:\Risk_Warehouse\Env\Casestudy_Env.sas7

Figure 4.8: The *Save Environment* window

If the *OK* button is clicked, a confirmation message that the risk environment has been saved, appears on the screen. This is illustrated in Figure 4.9.



Changes to environment Casestudy_Env were successfully saved to file C:\Risk_Warehouse\Env\Casestudy_Env.sas7bcat.

Figure 4.9: The *Confirmation* window

4.3 The creation of a new environment with *Proc Risk*

The starting point of this process is to create a new user-defined macro variable, named *RiskEnv*. This macro variable refers to the risk environment with name *Casestudy_Env* that is stored as a SAS catalog in Program Code 4.1. The physical location of the catalog is also specified.

```
%Let RiskEnv = &RiskPath\Env\Casestudy_Env;
```

In further program code, the risk environment named *Casestudy_Env* is referred to as *&RiskEnv*. Program Code 4.1 is used to create this new risk environment that is stored as a SAS catalog in the *Env* folder. The SAS catalog will also appear in the SAS library named *Env*. The *Environment* statement in *Proc Risk* is used to create new environments, open existing environments and save changes to environments.

Program Code 4.1: The creation of a new risk environment, *CaseStudy_Env*.

```
/*This procedure creates structures within SAS Risk Dimensions*/
Proc Risk;
/*The Environment statement is used to create a new environment with
name Casestudy_Env in the C:\Risk_Warehouse\Env folder */
Environment new = "&RiskEnv" ;
/*The  Environment  statements  is  used  to  save  the  Casestudy_Env
environment*/
Environment save;
Run;
```

If the risk management needs of the case study company had been more complex it may have been necessary to inherit information from other risk environments. Suppose the *Casestudy_Env* risk environment needs to inherit information from another risk environment, named *Physical_Asset_Env*, then the *Inherit* option in the *Environment* statement is used to inherit the information as illustrated in Program Code 4.2.

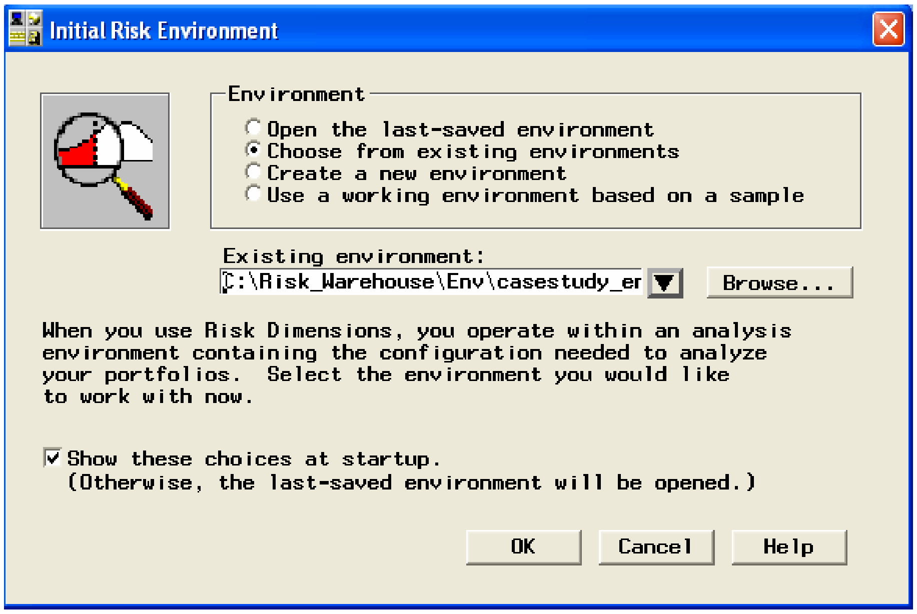
Program Code 4.2: The inheritance of another risk environment.

```
Proc Risk;
Environment new = "&RiskEnv" Inherit = "Physical_asset_env";
Environment save;
Run;
```

If Program Code 4.1 is executed the new risk environment is created. The GUI is used to view the newly created risk environment. The word *risk* is typed into the GUI icon and the ✓ button is clicked.

Figure 4.10: The GUI Icon

The *Initial Risk Environment* window opens. The Button: *Choose from existing environments* is selected. The *Browse* button is clicked and the name and physical location of the *Casestudy_Env* environment is specified.



Environment
- Open the last-saved environment
- [✓ Choose from existing environments]
- Create a new environment
- Use a working environment based on a sample

Existing environment: C:\Risk_Warehouse\Env\casestudy_en

When you use Risk Dimensions, you operate within an analysis environment containing the configuration needed to analyze your portfolios. Select the environment you would like to work with now.

[✓] Show these choices at startup.
(Otherwise, the last-saved environment will be opened.)

Figure 4.8: The *Initial Risk Environment* window

The *OK* button is clicked and the risk environment *Casestudy_Env* opens. The *Analysis* tree is viewed as illustrated in Figure 4.9.

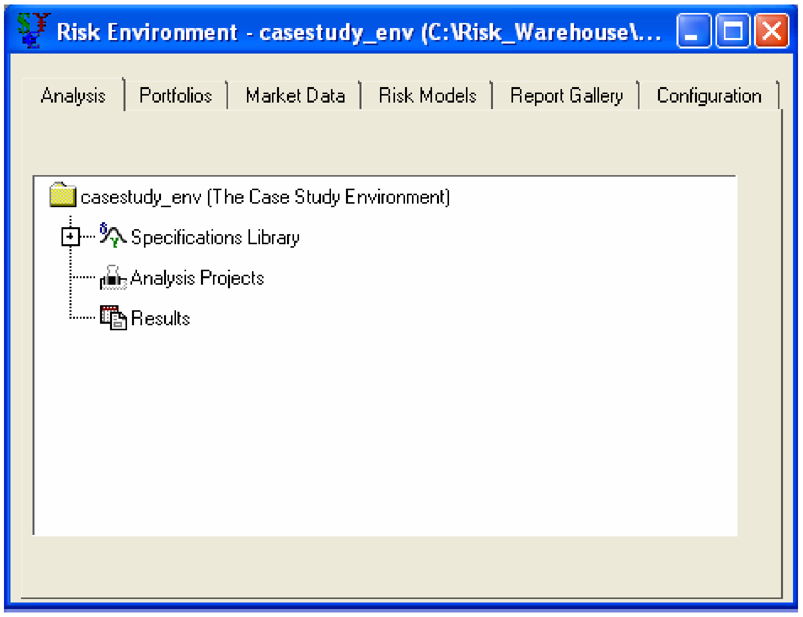


Figure 4.9: The *Analysis* tree of *Casestudy_Env*

In the subsequent SAS programs the procedure *Proc Risk* is used to create additional Risk Dimensions structures in this environment. An updated version of the risk environment is viewed each time, by repeating the steps described above.

4.4 Summary

The creation of a **risk environment** by using either the GUI or *Proc Risk* was discussed in this chapter. In the case study, the procedure *Proc Risk* will be used further to create a series of Risk Dimensions structures that form the risk management system. The GUI will be used on a regular basis to view these structures.

5

RISK DIMENSIONS VARIABLES

5.1 Introduction

Risk Dimensions variables are the first Risk Dimensions structures that are created in a risk environment. The variables are registered in a risk environment and are used to refer to data values that are needed in different stages of the risk management process. Some of the uses of the variables are discussed in the remainder of this section.

Some of the registered variables are used in the **valuation** of financial instruments. The following example illustrates the comparison between the valuation of an instrument in the real world and the steps that are necessary to implement the valuation process in SAS Risk Dimensions.

Suppose the financial instrument under consideration is an option on an equity. The **Black Scholes pricing function** $f(S_0, K, \sigma, r, t)$ is used to value each option in the portfolio where:

- $S_0 =$ the current market price of the underlying equity,
- $K =$ the strike price of the option,
- $\sigma =$ the annual volatility of the underlying equity,
- $r =$ the risk-free rate of interest and
- $t =$ the time to expiry.

Risk Dimensions variables are used to refer to data values that are used as input values in the pricing function $f(S_0, K, \sigma, r, t)$.

Consider the *Tradebook* SAS data set. The data values that are necessary in the calculation of the pricing function $f(S_0, K, \sigma, r, t)$ are stored in the columns with names *Strike*, *Enddate, OptType* and *Underlying*. For each of the columns a Risk Dimensions variable with the same name is registered. The rows (observations) of the data set are read one at a time. Each row represents one open position held in a financial instrument and contains a record of one observation for each column. For each row: the variable *Strike* contains the strike price (K) of the option, the data value contained in the variable *Enddate* minus the date of valuation equals the time to maturity (t) input value, and the data value in the variable *OptType* specifies if the option is a call or put and thus, which Black Scholes pricing method is to be used.

Suppose the data value in the *Underlying* column for a specific row, is *SOL*, then this means that the underlying equity is a Sasol equity. The current data values of the Sasol equity price and corresponding annualized volatility are contained in the *Market_History* SAS data set. Risk Dimensions variables with names *SOL* that refer to the price of the underlying equity (S_0) and *Vol_SOL* that refer to the annualized volatility of the underlying equity (σ), are subsequently registered. These variables link the data values in *Market_History* to the pricing function $f(S_0, K, \sigma, r, t)$.

In the risk management process, a yield curve is constructed. The curve consists of zero rates (risk-free rates of interest) with varying maturities. A Risk Dimensions variable is created for each zero rate. Examples of the variable names are *ZR_1_MTH*, *ZR_3_MTH*, *ZR_6_MTH*, *ZR_12_MTH* and *ZR_2_YEAR*. Linear interpolation is used to derive an estimate for the input parameter r (for a corresponding t) from the data values contained in these variables. It is important to note that the data values that are used in the zero rate variables like *ZR_1_MTH*, *ZR_3_MTH*, *ZR_6_MTH*, *ZR_12_MTH* and *ZR_2_YEAR* are not stored in SAS data sets. Some of the data values that are stored in SAS data sets are transformed into data values for these variables. The transformation process is discussed in Chapter 9.

In the risk management process, financial instruments are **grouped** or **classified** according to data values of registered variables, for example the variable *Book*. Each observation in the SAS data sets *Tradebook, Swapbook* and *Bondbook* has one data value for *Book*. The data values contained in *Book* are either *Com, Der* or *Int_Der*. The words are shortened descriptions for commodities, derivatives and interest rate derivatives. All the positions with a data value of *Com* are grouped together. This also holds for positions with data values of *Der* or *Int_Der.* The grouping or classification may be used in the creation of sub-portfolios.

Variables are also used in the calculation of **risk measures** like Value at Risk and the **modelling** of a time series of market data. These topics are discussed in detail in Chapter 10 and 8. Variables are also used to refer to calculated risk measure values and **references** that are used in pricing functions. Examples of these variables are discussed later in this chapter.

The objective of the rest of this chapter is to guide the user through the process of variable registration. Variables are defined by the graphical user interface (**GUI**) or by using ***Proc Risk*** in a SAS program. The GUI registers variables one at a time, so if the amount of variables to be registered is substantial, the effectiveness of this method declines. The method of using a SAS program with *Proc Risk* as a step, is the preferred method. This method is quite effective for registering large quantities of variables, since the program code necessary for the registration of an extra variable is not long. The GUI is used to view the variables that are registered in the risk environment (see Section 5.2.9). The variable structure of a risk environment is listed under the *Variables Definitions* option in the *Configuration* tree in the GUI and is illustrated in Figure 5.1.

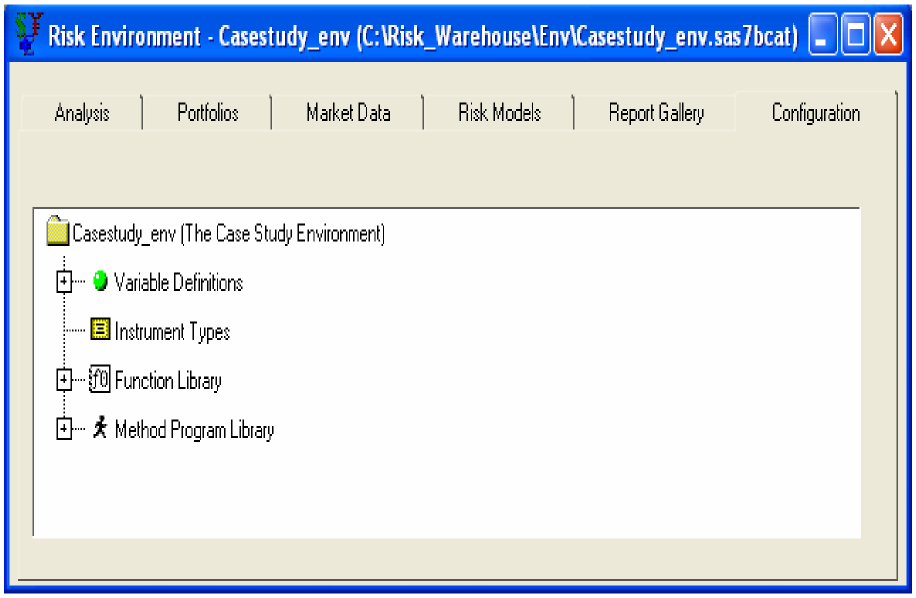


Figure 5.1: The *Configuration* tree of the *Casestudy_Env* risk environment

If the *Variable Definitions* option is selected, the different kinds of variables are shown:

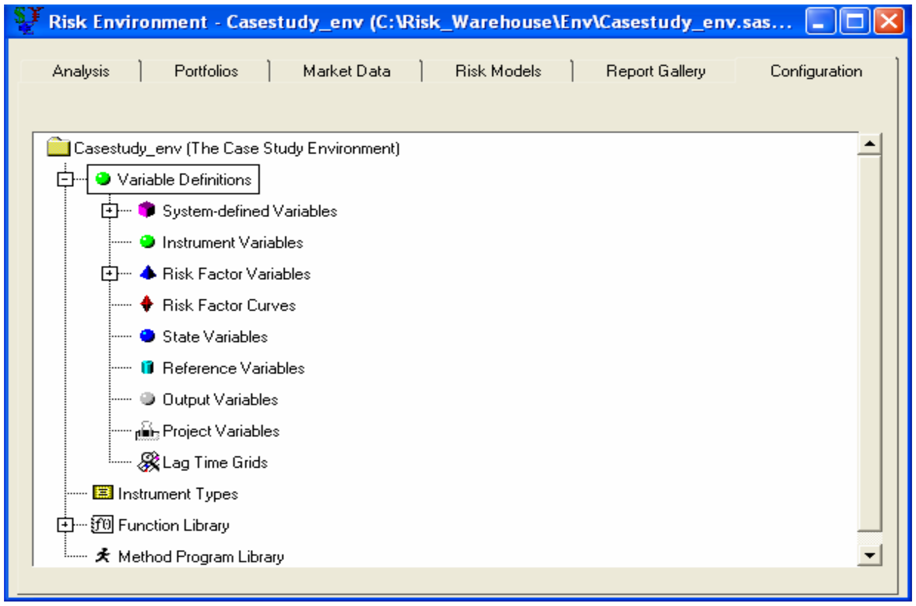


Figure 5.2: The *Variable Definitions* option in the *Configuration* tree.

All variables that have been registered in the risk environment are viewed in the *Configuration* tree of the GUI. The variables are listed under the appropriate variable types, that were assigned during the registration process.

It is important to determine **which variables** need to be registered and furthermore, **how** these variables need to be registered. A variable can be registered in many different ways. The use of the variable in the risk environment is the deciding factor in the way it is registered. The different kinds of variables that are available in each risk environment are discussed in detail in Section 5.2. The registration process is illustrated in terms of the case study. An **alternative** variable registration method, data-driven registration, is discussed in Chapter 6.

Consider the case study again. The variables that need to be registered in the *Casestudy_Env* risk environment are broadly divided into two groups:

1. The relevant variables that refer to data values in the *Tradebook, Bondbook*, *Swapbook*, *Logreturns*, *Yieldcurve_data, Market_history* and *Scenariodata* SAS data sets.

2. The variables that are necessary for the risk management system, but refer to values that are not available in these SAS data sets.

The variables in group 1 are declared either by using *Proc Risk* in a SAS program or data-driven registration (see Chapter 6). The variables contained in group 2 can only be declared by using *Proc Risk.*

Table 5.1 contains the name, description, purpose and data type of the variables that need to be registered for the case study. An example and the group specification for each variable are also included.

Table 5.1: List of variables that are registered in the *Casestudy_Env* environment

Variable name	**Description**	**Purpose of variable**	**Data type**	**Example**	**Group**
Currency	Currency of interest rate	Used in valuation of instruments	Currency code	ZAR	1
InstType	Type of instrument	Classifies an instrument	Character	Equity, Option, Future	1
Instid	Instrument Identification	Unique name for instrument	Character	SOL_001	1
Holding	Amount of instruments in position taken	Used in valuation of portfolio value	Numeric	10000	1
Shortposition	Short position of position held	Indicates whether position held, is long or short	Numeric	0 (long) 1 (short)	1
Strike	Strike price of option	Used in valuation of an option	Numeric	100	1
Enddate	Redemption day of an option or future	Used in valuation of an option or a future	SAS date value	16624	1
OptType	Type of Option (EC, AC EP, AP)	Used to contain full information about the type of option	Character	EC (European Call)	1
Input_OptType	Type of Option (Call or Put)	Used in the valuation of an option	Character	Call	2
Book	Book of commodities, derivatives and interest rate derivatives	Classify between different groups of instruments	Character	Der (Derivative)	1
Premium	Price paid for instrument	Used to calculate profit of position	Numeric	34.5	1
Sector	Equity sector	Classify between equity sectors	Character	Fin (Financials)	1
Underlying	Underlying asset of the derivative	Used in valuation of an option or a future	Character	SOL	1
Marketprice	Market price of equity	Used in the valuation of an equity	Character	ASA	1
Contractprice	Contract price of future	Used in the valuation of a future	Numeric	87.2	1
Daily_profit	Value of instrument minus premium paid	Used to calculate profit of position	Numeric	34.2	2
Coupfreq	Frequency of bond coupon payments	Used in valuation of a bond	Numeric	6	1
Fixrate	Fixed interest rate used in swap payments	Used in valuation of an interest rate swap	Numeric	0.08	1
Notional	Notional amount used in bond and swap positions	Used in valuation of a swap and a bond	Numeric	1000 000	1

Variable name	Description	Purpose of variable	Data type	Example	Group
Fromdate	Date used for exchange dates in interest rate swaps	Used in valuation of an interest rate swap	SAS Date Value	16517	1
Ftr_name	Name of floating interest rate	Used in valuation of an interest rate swap	Character	JB_6_MTH	1
RcveType	Indicator of receiving payments (floating or fixed)	Used in valuation of an interest rate swap	Character	Floating	1
Coupon	Coupon rate of bonds	Used in valuation of a bond	Numeric	0.13	1
Red_Amount	Redemption amount of bonds	Used in valuation of a bond	Numeric	100	1
ZR_1_MTH	One month zero rate	Used in yield curve construction	Numeric	0.075	2
ZR_3_MTH	Three month zero rate	Used in yield curve construction	Numeric	0.077	2
ZR_6_MTH	Six month zero rate	Used in yield curve construction	Numeric	0.078	2
ZR_12_MTH	Twelve month zero rate	Used in yield curve construction	Numeric	0.08	2
ZR_18_MTH	Eighteen month zero rate	Used in yield curve construction	Numeric	0.081	2
ZR_2_YEAR	Two year zero rate	Used in yield curve construction	Numeric	0.082	2
ZR_30_MTH	Twenty four month zero rate	Used in yield curve construction	Numeric	0.083	2
ZR_3_YEAR	Three year month zero rate	Used in yield curve construction	Numeric	0.084	2
ZR_42_MTH	Thirty month zero rate	Used in yield curve construction	Numeric	0.085	2
ZR_4_YEAR	Four year zero rate	Used in yield curve construction	Numeric	0.086	2
ZR_54_MTH	Fifty four month zero rate	Used in yield curve construction	Numeric	0.087	2
ZR_5_YEAR	Five year zero rate	Used in yield curve construction	Numeric	0.088	2
ZR_66_MTH	Sixty six month zero rate	Used in yield curve construction	Numeric	0.089	2
ZR_6_YEAR	Six year zero rate	Used in yield curve construction	Numeric	0.0895	2
ZR_78_MTH	Seventy eight month zero rate	Used in yield curve construction	Numeric	0.09	2
ZR_7_YEAR	Seven year zero rate	Used in yield curve construction	Numeric	0.091	2
ZR_90_MTH	Ninety month zero rate	Used in yield curve construction	Numeric	0.09	2
ZR_8_YEAR	Eight year zero rate	Used in yield curve construction	Numeric	0.094	2
ZR_102_MTH	One hundred and two month zero rate	Used in yield curve construction	Numeric	0.09	2

Variable name	Description	Purpose of variable	Data type	Example	Group
ZR_9_YEAR	Nine year zero rate	Used in yield curve construction	Numeric	0.096	2
ZR_114_MTH	One hundred and fourteen month zero rate	Used in yield curve construction	Numeric	0.09	2
ZR_10_YEAR	Ten year zero rate	Used in yield curve construction	Numeric	0.098	2
JB_6_MTH	JIBAR 6 Month yield	Used as a floating rate for interest rate swaps	Numeric	0.078	1
Prin1	Contains the first principal component	Used in risk factor modelling	Numeric	-2.57392	2
Prin2	Contains the second principal component	Used in risk factor modelling	Numeric	1.2453	2
Prin3	Contains the third principal component	Used in risk factor modelling	Numeric	-3.14143	2
ASA	ABSA equity price	Used in valuation of instruments	Numeric	36.43	1
SOL	Sasol equity price	Used in valuation of instruments	Numeric	89.31	1
ISC	Iscor equity price	Used in valuation of instruments	Numeric	35.32	1
SLM	Sanlam equity price	Used in valuation of instruments	Numeric	7.89	1
OML	Old Mutual equity price	Used in valuation of instruments	Numeric	12.21	1
AGL	Anglo equity price	Used in valuation of instruments	Numeric	133.50	1
ASA_Vol	Volatility of Absa equity price	Used in valuation of an equity option	Numeric	0.31	1
SOL_Vol	Volatility of Sasol equity price	Used in valuation of an equity option	Numeric	0.28	1
ISC_Vol	Volatility of Iscor equity price	Used in valuation of an equity option	Numeric	0.24	1
SLM_Vol	Volatility of Sanlam equity price	Used in valuation of an equity option	Numeric	0.23	1
OML_Vol	Volatility of Old Mutual equity price	Used in valuation of an equity option	Numeric	0.35	1
AGL_Vol	Volatility of Anglo equity price	Used in valuation of an equity option	Numeric	0.27	1
UNSTD_1_MTH	One month rate	Used in yield curve construction	Numeric	0.067	1
UNSTD_3_MTH	Three month rate	Used in yield curve construction	Numeric	0.071	1
UNSTD_6_MTH	Six month rate	Used in yield curve construction	Numeric	0.073	1
UNSTD_12_MTH	Twelve month rate	Used in yield curve construction	Numeric	0.075	1

Variable name	Description	Purpose of variable	Data type	Example	Group
UNSTD_18_MTH	Eighteen month rate	Used in yield curve construction	Numeric	0.076	1
UNSTD_2_YEAR	Two year rate	Used in yield curve construction	Numeric	0.08	1
UNSTD_30_MTH	Twenty four month rate	Used in yield curve construction	Numeric	0.081	1
UNSTD_3_YEAR	Three year month rate	Used in yield curve construction	Numeric	0.082	1
UNSTD_42_MTH	Thirty month rate	Used in yield curve construction	Numeric	0.083	1
UNSTD_4_YEAR	Four year rate	Used in yield curve construction	Numeric	0.084	1
UNSTD_54_MTH	Fifty four month rate	Used in yield curve construction	Numeric	0.085	1
UNSTD_5_YEAR	Five year rate	Used in yield curve construction	Numeric	0.086	1
UNSTD_66_MTH	Sixty six month rate	Used in yield curve construction	Numeric	0.087	1
UNSTD_6_YEAR	Six year rate	Used in yield curve construction	Numeric	0.0885	1
UNSTD_78_MTH	Seventy eight month rate	Used in yield curve construction	Numeric	0.09	1
UNSTD_7_YEAR	Seven year rate	Used in yield curve construction	Numeric	0.091	1
UNSTD_90_MTH	Ninety month rate	Used in yield curve construction	Numeric	0.09	1
UNSTD_8_YEAR	Eight year rate	Used in yield curve construction	Numeric	0.094	1
UNSTD_102_MTH	One hundred and two month rate	Used in yield curve construction	Numeric	0.09	1
UNSTD_9_YEAR	Nine year rate	Used in yield curve construction	Numeric	0.097	1
UNSTD_114_MTH	One hundred and fourteen month rate	Used in yield curve construction	Numeric	0.098	1
UNSTD_10_YEAR	Ten year rate	Used in yield curve construction	Numeric	0.101	1

5.2 The different kinds of variables

5.2.1 General

As mentioned earlier, the purpose and use of a variable in a risk environment, determines the way in which it is registered.

Each variable is registered as one of the following nine kinds of Risk Dimensions variables:

1. **System defined variables** are a list of variables created automatically by SAS, for example *Currency*.
2. **Instrument variables** contain all the available information about financials instruments when the positions are taken, for example *Premium*.
3. **Risk factor variables** are used to refer to market information, for example ASA (the ABSA equity price).
4. **Risk factor curves** are arrays of risk factor variables. An example is a yield curve that consists of zero rates with different maturities.
5. **Output variables** are used to carry a value computed in a pricing method to the output data sets, for example *Daily_profit.*
6. **Reference variables** are used to link market information and portfolio information together during pricing methods. An example of a variable of this kind is *Price*.
7. **Lag time grids** are used when a lagged value of a risk factor is needed during a pricing method. An example is the value of the risk factor variable *JB_6_MTH* four months ago, that is necessary in the valuation of an interest rate swap.
8. **Project variables** are an advanced topic of Risk Dimensions. They are used to pass information between user-defined functions.

9. **State variables** are also an advanced feature of SAS Version 9 that are used to pass market state information between user-defined functions.

Project and State variables are not discussed in this document.

Each variable is registered or declared as one of the nine kinds of variables above. In addition, attributes are registered for each variable that elaborates the way in which it is declared. Certain attributes are compulsory for certain kinds of variables, whilst various optional attributes also exist.

Irrespective of which kind of variable is considered, the following three attributes need to be declared:

- *Name*,
- *Type* and
- *Role*.

Each variable must have a valid SAS **name** with a maximum length of 32 characters. Any combination of characters, numbers and underscores are allowed in the name. The data type of the values of the variable are defined in the attribute ***type*****.** Character and numeric data types are allowed for all the kinds of variables. Additional data types are available for some kinds of variables. The ***role*** attribute is very useful in certain risk analyses. Special roles are assigned to some variables, but the majority of variables are assigned a generic role. The range of **optional attributes** differs for each kind of variable. More information will be included in the separate variable discussions.

The seven different kinds of variables that are used most frequently in Risk Dimensions environments, are discussed in detail in Sections 5.2.2 to 5.2.8. The case study is used to illustrate the use of each kind of variable.

5.2.2 System defined variables

Not all the variables that are used in the risk environment are defined by the user. **System defined variables** are variables that are automatically created by Risk Dimensions. A new risk environment already contains system defined variables. The variables are used in the valuation and grouping of financial instruments. It is also used for the computed values of the pricing functions of the instruments. Background information about the most frequently used system defined variables is discussed in this section.

Consider the three SAS data sets named *Tradebook*, *Swapbook* and *Bondbook*. These data sets contain information about the open positions held in financial instruments. The columns with names *InstType* and *Instid* are used to contain information about the type of financial instrument, as well as, a unique identification. Corresponding system defined variables with the same names exist. The variables are:

InstType

This variable provides a link to the Risk Dimensions structure, named an **instrument type**. The names of the financial instruments (instrument types) in the portfolio are stored as data values in this variable. Instrument types are discussed in detail in Section 7.5. Examples of data values contained in the variable, are *equity*, *option* and *future*.

InstID

This variable refers to an identification of the instrument held in the open position. An example is *SOL_001*, which means that the instrument is the *Sasol* equity with an identification number of *001*.

As mentioned above, a Risk Dimensions structure, called an instrument type is created for each type of financial instrument during the creation of the risk management system. A list of variables is specified for each instrument type during the creation of the structure. *Instid* and *InstType* may not be included in this list.

The following system defined variables are used to provide information about the financial instruments in the portfolio and may be included in the variable list of an instrument type:

Collateral

It is a numeric flag that identifies instruments that are held as collateral, but are not owned. Instruments with a non-zero value for *Collateral* are only used in credit exposure analysis and are not included in the portfolio file.

CounterpartyID

It is a character variable that contains information about the counterparty of the position held.

Currency

Instruments can be valued in different currencies. This variable is used to translate the calculated values in a foreign currency, into the corresponding value of the reporting currency. *Currency* is a character variable of length three and must contain a valid three letter ISO currency code, for example *ZAR*.

Holding

The *Holding* variable refers to the number of instruments held in the position.

Maturitydate

Maturitydate refers to the maturity date of the instrument held in the position and must be a valid SAS date value.

NetSetID

The variable is used to control the netting of instruments in credit exposure calculations.

ShortPosition

It is a numeric flag that indicates a short position in the instrument. A non-zero value indicates a short position.

The following system defined variables are used to contain calculated values. The calculated values are obtained, for example, from the valuation of an instrument.

*_VALUE_*

The computed instrument value is assigned to this variable. The variable *_VALUE_* is a required variable in each pricing method.

BaseCase

It is a zero-one numeric flag that indicates whether the base-case mark-to-market valuation is being computed.

BaseDate

This variable refers to the date of the base-case mark-to-market.

The following system defined variables contain calculated values and may appear in any output data set, created by the execution of a risk analysis, for example Value at Risk.

Value

The displayed value is the value of the position held. It is the value of one instrument, multiplied by the value in the variable *Holding*, taking short positions and currency calculations into account.

NatValue

It is the value of the open position in the native currency. The value of one instrument is multiplied by the value in the *Holding* variable, taking short positions also into account.

FX_Rate

The value of the foreign exchange rate that is used to translate the value of the instrument to the reporting currency, is used in this variable.

*_date_*

This variable refers to the date of valuation, on which the instruments are priced and the portfolio value is calculated. Although this variable is a system defined variable, it is listed in the *Risk Factor Variables* option in the *Configuration* tree of the GUI.

*_Cashflow_*

The variable is used in the valuation of financial instruments. It is discussed in more detail in Section 7.4.3.

Other system defined variables are: *Altype*, *Alposition*, *Inst_Number*, *InstSource*, *Insttotal*, *NumeraireCurrency*, *SimulationCase*, *SimulationCategory*, *SimulationHorizon*, *SimulationInterval*, *SimulationMode*, *SimulationReplication*, *SimulationTime*, *Cfbucket*, *Cumbucket*, *Exposure*, *MacaulayCnvx*, *MacaulayDur*, *Modified_Cnvx*, *Modified_Dur*, *Return*, *_Cash_*, *_Cumcf_*, *_Type_, _Name_,* *_CValue_, _NValue_, _CFCUR_, _CFDate_, _CFAMNT_, Cash_Flows* and *Quad_Value*. In depth knowledge about these variables is not essential in the case study risk management system. Hence, these variables are not discussed in this document. In a different case study problem it might be necessary to study these variables in more detail.

Case study

All the variables in Table 5.1 must be declared or registered as some kind of Risk Dimensions variable. The first step of the registration process is to find all the variables that are already declared as system defined variables. There is no further need to declare them.

The following variables are already declared as system defined variables: *InstType, Instid, Currency, Holding* and *Shortposition.*

5.2.3 Instrument variables

Variables that refer to information that define some of the characteristics of the financial instruments in the portfolio are declared as **instrument variables**.

If a position is taken in an option, the following variables contain information regarding this position: *Strike* (strike price of the option), *Enddate* (date of expiry) and O*ptType* (the type of option: European call, American call or European put). The data value in the variable *Book* namely *Der* classifies the option as a derivative. All these variables mentioned and others are declared as instrument variables.

The ***Declare Instvars*** statement is used to declare instrument variables with the necessary attributes in program code. The general form of this statement follows:

Declare Instvars = (Name Type Length Role “Optional Attributes”);

A valid SAS name must be specified in the ***name*** attribute.

One of the following data **types** is specified for each instrument variable:

• *Num*	numeric values,
• *Char*	character values,
• *Date*	SAS Date Values and
• *Array*	*SAS* arrays*.*

The ***length*** attribute is only declared if the ***type*** attribute is defined as *char*. It specifies the length of the largest data value in the variable. The ***role*** attribute defines the use of the instrument variable. If the variable is used to value an instrument, the variable is defined as an **instrument attribute** (*role* = *var*). If the instrument variable is used to group or classify instruments, the variable is specified as a **classification variable** (*role* = *class*).

If an option is considered again, then variables like *Strike*, *Enddate* and O*ptType* are necessary in the pricing method and are hence defined as instrument attributes. The data values in the variable *Book* are used to group or classify the instruments in the portfolio. Hence, *Book* is defined as a classification variable.

The following **optional attributes** may be specified for an instrument variable in the *Declare Instvars* statement:

Label = label

A descriptive character label may be specified for the variable.

Drop = variable-name

The variable specified in variable name is not included in the output data sets (see Section 10.7). Output data sets are created by the execution of risk analyses such as Value at Risk.

Format = “Format”

A SAS format, that is used to display the variable values in a certain format, may be specified.

Group = “Group”

A group is specified for the variable. Variables are displayed by the group attributes in the *Configuration* tree of the GUI. If no group is specified, this attribute is set equal to the value of the *role* attribute of the variable.

Postvar = value

The value specified in this attribute indicates that the value of the instrument variable is calculated by a *Postvar* method program. More information about *Postvar* method programs is available in Section 7.4.

Case study

The following variables in Table 5.1 are registered as instrument variables:
Strike, Enddate, OptType, Input_OptType, Book, Premium, Sector, Underlying, Marketprice, Contractprice, *Coupfreq*, *Fixrate*, *Notional*, *Ftr_Name*, *Fromdate*, *Rcvetype,* and *Red_Amount*.

Variables that are registered as instrument variables are used either to:

- Value instruments (*role* = *var*) or
- Group or classify instruments (*role* = *class*)

The variables *Strike*, *Enddate* and *Underlying* are used in the valuation of the options. The data values in the variable *OptType* specifies if the option is a European call (*EC*), an American call (*AC*) or a European put (*EP*). The variable *Input_OptType* specifies if the option is a call or a put. *Marketprice* is used in the valuation of equities, while the variables *Contractprice* and *Underlying* are used in the valuation of futures. The variables *Coupfreq*, *Notional,* and *Red_Amount* are used in the valuation of government bonds. *Fixrate*, *Notional*, *Ftr_Name*, *Fromdate* and *Rcvetype* are used in the valuation of interest rate swaps. The *Premium* variable is used in the calculation of profit. The variables *Sector* and *Book* may be used to classify or group instruments in the portfolio. Program Code 5.1 illustrates the registration of instrument variables in the *Casestudy_Env* risk environment, by using the procedure *Proc Risk*.

Program Code 5.1: The registration of instrument variables.

```
/*This procedure creates structures within SAS Risk Dimensions*/
Proc Risk;
/*The Casestudy_Env risk environment is opened*/
Environment open = "&RiskEnv";
/*The following statement declares instrument variables within the
Casestudy_Env environment*/
Declare Instvars =
   (Strike         num     var   Label = "Strike price of option",
    Enddate        date    var   Label = "Date of expiry",
    Premium        num     var   Label = "Cost of instrument",
    Underlying     char 12 var   Label = "Underlying asset",
    Marketprice    char 10 var   Label = "The equity price",
    Contractprice  num     var   Label = "Contract price of Future",
    OptType        char 6  var   Label = "Type of option EC/AC/EP/AP",
    Input_OptType  char 6  var   Label = "Option type in pricing
                                          method",
    Sector         char 12 class Label = "The Equity Sectors",
    Coupfreq       num     var   Label = "Coupon Frequency",
    Fixrate        num     var   Label = "Fixed Interest Rate in swap",
    Notional       num     var   Label = "Nominal amount of interest
                                          rate swap",
    Ftr_name       char 12 var   Label = "Floating Interest Rate
                                          Name",
    Fromdate       date    var   Label = "Previous exchange date of
                                          swap",
    Rcvetype       char 12 var   Label = "Indicator of receiving swap",
    Coupon         num     var   Label = "Coupon rate of bond",
    Red_Amount     num     var   Label = "The Redemption amount of
                                          the bond",
    Book           char 12 class Label = "The type of trade book" );
```

Program Code 5.1 continues …

```
/*The Casestudy_Env is saved, with the added instrument variables
declared above*/
Environment save;
Run;
```

5.2.4 Risk factor variables

The historical information of **market variables**, for example, equity prices and interest rates is stored in the SAS data set *Market_History* in the case study. Market information has also been used to create the SAS data sets *Logreturns* and *Yieldcurve_data.* The logarithm of the daily return between equity prices has been calculated in *Logreturns*. The JIBAR Rates and swap rates were used to calculate zero rates for different maturities. The calculated zero rates are contained in *Yieldcurve_data*. The calculations were done **outside** of the SAS environment in Microsoft Excel. Thus, the columns in these SAS data sets contain **observed** or **derived** market information. The column names of these data sets are registered as **risk factor variables**.

It is further, possible to use the data contained in the SAS data sets to **derive** new data values that are used in the risk management system. These values are derived or calculated **in** the Risk Dimensions. It follows that risk factor variables also have to be created to contain these new data values. Examples of risk factor variables that are first assigned values in Risk Dimensions, are discussed in Program Code 5.2 later in this section.

Market variables are used to calculate the value of the financial instruments in the portfolio. A risk factor variable may be viewed as the Risk Dimensions variable that corresponds to a market variable. All the different interest rates, instrument prices and derived variables are declared as risk factor variables in the risk environment. These risk factor variables are used and referred to in pricing methods and risk analysis calculations.

The ***Declare Riskfactors*** statement is used to declare or register risk factor variables in a risk environment and has the following general form:

Declare Riskfactors = (Name Type Role "Optional Attributes");

One of the following data **types** is specified for each risk factor variable:

• *Num*	numeric values
• *Date*	SAS date values

The **role** of a risk factor variable determines how it can be used in risk analyses. One of the following role attributes is required for a risk factor variable:

• *Var*	specifies an undefined or generic role for the risk factor variable.
• *IR*	specifies that the risk factor variable is an interest rate. This makes the declaration of the *Currency* and *Maturity* attributes compulsory.
• *FX*	specifies that the risk factor variable is an exchange rate between two currencies. This makes the declaration of the *Currency, Fromcur* and *Tocur* attributes compulsory.
• *FX_Spot*	specifies that the risk factor variable is a spot exchange rate between two currencies. The attributes *Currency, Fromcur and Tocur* must be declared.
• *Volatility*	specifies that the risk factor variable is the volatility of another risk factor variable. The *Basevar* attribute has to be specified.

Optional attributes

The following optional attributes may be specified for each risk factor variable in the *Declare Riskfactors* statement:

Basevar = risk-factor-variable-name

The attribute is used when the *role* attribute is set to *Volatility*. The name of the risk factor variable that is provided with a volatility estimate, is specified.

Category = category-list

The values specified in the *category-list* option are user-defined. This attribute is used in the calculation of marginal and conditional Value at Risk (VaR). Marginal VaR analysis is performed by fixing all the risk factor variables in the category and changing all the risk factor variables that are not in the category. Conditional VaR analysis is performed by fixing all the risk factor variables that are not in the category and changing all the risk factor variables in the category. The use of the attribute is discussed in more detail in Chapter 10.

Currency = currency-code

This attribute is used only in the declaration of risk variables with interest rate roles. It specifies the currency for an interest rate variable.

Format = format

The SAS format that is used to display values of the risk factor variable that is declared, is specified in this attribute.

Fromcur = currency-code

The currency that is converted by the foreign exchange rate is specified in this attribute.

Label = variable-label

A descriptive label may be specified for the risk factor variable.

Laggrid = timegrid-name

The name of the lag time grid structure that contains lagged values of the risk factor variable declared, is specified. Time grids are discussed in detail in Section 5.2.8.

Maturity = number

A maturity value is assigned to the risk factor variable. This value is measured in the maturity unit that is specified in the *Munit* attribute.

Munit = maturity-unit

A time unit is assigned to the maturity value as specified in the *Maturity* attribute.

Mlevel = measurement-level

The measurement level of the risk factor variable is specified as either *interval* or *ratio*. The default value is *ratio* and may only contain values greater or equal to nought.

Refmap = Reference-variable - Reference-Key-Value

Reference variables are linked with reference key values by this attribute. This is essential for use in pricing methods. The use of this attribute is discussed in more detail in Section 5.2.7.

Tocur = currency-code

The currency that is created by the foreign exchange rate is specified in this attribute.

Group = variable group

A group is assigned to the risk factor variable. Risk factor variables are grouped in the GUI according to the value in the *group* attribute. If the group attribute is not explicitly specified, the value is set equal to the value of the *role* attribute of the risk factor variable.

Case study

All the market variables containing data values in the *Market_History, Logreturns* and *Yieldcurve_data* data sets are declared as risk factor variables in the *Casestudy_Env* risk environment. Other market variables that are derived by transforming market variables in Risk Dimensions, and that are needed later on in the risk analysis process, are also declared as risk factor variables. Program Code 5.2 is used to declare the risk factor variables in the *Casestudy_Env* risk environment.

Program Code 5.2: The registration of risk factor variables.

```
Proc Risk;
Environment open = "&RiskEnv";
/*The following statement declares the risk factor variables in the
Casestudy_Env risk environment*/
Declare Riskfactors =
(JB_6_MTH  num ir Label = "Floating Rate of Swap" Laggrid = grid1
           Refmap = (floatingrate = "JB_6_MTH") Currency = ZAR
           Maturity = 0.5 year Category = "IR",
/*The risk factor variable above is declared with a numeric data type
(type = num) and an interest rate role (role = ir). The Maturity
attribute is specified as 6, followed by the maturity unit, namely
month. The Currency attribute is compulsory, because an interest rate
role is specified. Label, Laggrid, Refmap, and Category are optional
attributes that are also specified.  */
 SOL       num  var  Group = "Commodity" Label = "Sasol Equity Price"
           Refmap = (price = "SOL") Category = "Commodity"
           Mlevel = interval,
 ASA       num  var  Group = "Commodity" Label = "Absa Equity Price"
           Refmap = (price = "ASA") Category = "Commodity"
           Mlevel = interval,
 SLM       num  var  Group = "Commodity" Label = "Sanlam Equity Price"
           Refmap = (price = "SLM") Category = "Commodity"
           Mlevel = interval,
 ISC       num  var  Group = "Commodity" Label= "Iscor Equity Price"
           Refmap = (price = "ISC") Category = "Commodity"
           Mlevel = interval,
 OML       num  var  Group = "Commodity" Category = "Commodity"
           Label = "Old Mutual Equity Price" Refmap = (price = "OML")
           Mlevel = interval,
```

Program Code 5.2 continues …

```
AGL         num  var  Group = "Commodity" Label= "Anglo Equity Price"
            Refmap = (price = "AGL") Category = "Commodity"
            Mlevel = interval,
/*The six risk factor variables above are declared as having a numeric
data type (type = num) and a generic variable role (role = var). The
Refmap attribute is discussed in more detail in Section 5.2.7. The
Group, Mlevel, Category and Label attributes are also specified. */
 Vol_AGL    num volatility Group = "Volatility" Basevar = AGL
            Label = "Volatility in Anglo Equity Price"
            Category = "Volatility",
 Vol_ASA    num volatility Group = "Volatility" Basevar = ASA
            Label = "Volatility in Absa Equity Price"
            Category = "Volatility",
 Vol_ISC    num volatility Group = "Volatility" Basevar = ISC
            Label = "Volatility in Iscor Equity Price"
            Category = "Volatility",
 Vol_OML    num volatility Group = "Volatility" Basevar = OML
            Label = "Volatility in Old Mutual Equity Price"
            Category = "Volatility",
 Vol_SLM    num volatility Group= "Volatility" Basevar = SLM
            Label = "Volatility in Sanlam Equity Price"
            Category = "Volatility",
 Vol_SOL    num volatility Group = "Volatility" Basevar = SOL
            Label = "Volatility in Sasol Equity Price"
            Category = "Volatility",
/*The six variables above are declared as having a numeric data type
(type = num) and a volatility role (role = volatility). Due to the
volatility role, the Basevar attribute is also specified. The optional
attributes Group and Label are also defined.*/
 ZR_1_MTH   num  ir  Label = "1 month Zero Rate" Currency = ZAR
            Maturity = 1 month Category = "IR",
 ZR_3_MTH   num  ir  Label = "3 month Zero Rate" Currency = ZAR
            Maturity = 3 month Category = "IR",
 ZR_6_MTH   num ir Label = "6 month Zero Rate" Currency = ZAR
            Maturity = 6 month Category = "IR",
 ZR_12_MTH num ir Label = "12 month Zero Rate" Currency = ZAR
            Maturity = 12 month Category = "IR",
 ZR_18_MTH num ir Label = "18 month Zero Rate" Currency = ZAR
            Maturity = 1.5 year Category = "IR",
 ZR_2_YEAR num ir Label = "2 year Zero Rate" Currency = ZAR
            Maturity = 2 year Category = "IR",
 ZR_30_MTH num ir Label = "2.5 year Zero Rate" Currency = ZAR
            Maturity = 2.5 year Category = "IR",
 ZR_3_YEAR num ir Label = "3 year Zero Rate" Currency = ZAR
            Maturity = 3 year Category = "IR",
 ZR_42_MTH num ir Label = "3.5 year Zero Rate" Currency = ZAR
            Maturity = 3.5 year Category = "IR",
 ZR_4_YEAR num ir Label = "4 year Zero Rate" Currency = ZAR
            Maturity = 4 year Category = "IR",
 ZR_54_MTH num ir Label = "4.5 year Zero Rate" Currency = ZAR
            Maturity = 4.5 year Category = "IR",
 ZR_5_YEAR num ir Label = "5 year Zero Rate" Currency = ZAR
            Maturity = 5 year Category = "IR",
```

Program Code 5.2 continues …

```
ZR_66_MTH num ir Label = "5.5 year Zero Rate" Currency = ZAR
           Maturity = 5.5 year Category = "IR",
 ZR_6_YEAR num ir Label = "6 year Zero Rate" Currency = ZAR
           Maturity = 6 year Category = "IR",
 ZR_78_MTH num ir Label = "6.5 year Zero Rate" Currency = ZAR
           Maturity = 6.5 year Category = "IR",
 ZR_7_YEAR num ir Label = "7 year Zero Rate" Currency = ZAR
           Maturity = 7 year Category = "IR",
 ZR_90_MTH num ir Label = "7.5 year Zero Rate" Currency = ZAR
           Maturity = 7.5 year Category = "IR",
 ZR_8_YEAR num ir Label = "8 year Zero Rate" Currency = ZAR
           Maturity = 8 year Category = "IR",
ZR_102_MTH num ir Label = "8.5 year Zero Rate" Currency = ZAR
           Maturity = 8.5 year Category = "IR",
 ZR_9_YEAR num ir Label="9 year Zero Rate" Currency = ZAR
           Maturity = 9 year Category = "IR",
ZR_114_MTH num ir Label="9.5 year Zero Rate" Currency = ZAR
           Maturity = 9.5 year Category = "IR",
ZR_10_YEAR num ir Label="10 year Zero Rate" Currency = ZAR
           Maturity = 10 year Category = "IR",
/*The twenty two variables above are used to construct a yield curve.
They are declared as having a numeric data type (type = num) and an
interest rate role (role = ir).Due to the interest rate role, the
Currency and Maturity attributes are also specified. The optional
attributes Label and Category are also defined. Data values are assigned
to these variables in Risk Dimensions. This is discussed in Chapter 9.*/
 UNSTD_1_MTH  num  var  Label = "1 month Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_3_MTH  num var Label = "3 month Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_6_MTH  num var Label = "6 month Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_12_MTH num var Label = "12 month Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_18_MTH num var Label = "18 month Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_2_YEAR num var Label = "2 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_30_MTH num var Label = "2.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_3_YEAR num var Label = "3 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_42_MTH num var Label = "3.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_4_YEAR num var Label = "4 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_54_MTH num var Label = "4.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_5_YEAR num var Label = "5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_66_MTH num var Label = "5.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_6_YEAR num var Label = "6 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
```

Program Code 5.2 continues …

```
 UNSTD_78_MTH num var Label = "6.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_7_YEAR num var Label = "7 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
 UNSTD_90_MTH num var Label = "7.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
UNSTD_8_YEAR num var Label = "8 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
UNSTD_102_MTH num var Label="8.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
UNSTD_9_YEAR num var Label = "9 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
UNSTD_114_MTH num var Label = "9.5 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
UNSTD_10_YEAR num var Label = "10 year Unstd Zero Rate"
              Category = "Var" Mlevel = interval,
/*The twenty two variables above are used in the construction of a yield
curve.  Although the starting values are interest rates, the variable
may later contain negative values after transformation has taken place.
Thus they are declared as having a numeric data type (type = num) and an
unspecified role (role = var).The Mlevel, Label and Category attributes
are also defined for each variable.*/
 Prin1     num var Mlevel = interval
           Label = "The First Principal Component",
 Prin2     num var Mlevel = interval
           Label = "The Second Principal Component",
 Prin3     num var Mlevel = interval
           Label = "The Third Principal Component");
/*The three variables above are used to refer to the values of the first
three principal components, that are used in the risk management system.
They are declared as having a numeric data type (type = num) and an
unspecified role (role = var). The Mlevel and Label attributes are also
defined for each variable.*/
Environment save;
/*The Casestudy_Env is saved, with the added risk factor variables that
were declared above*/
Run;
```

5.2.5 Risk factor curves or arrays

Risk factor variables were discussed and declared in the previous section. Risk factor variables may be grouped together to form another Risk Dimensions structure called a **risk factor curve** or a **risk factor array**. A risk factor curve is also a special type of a SAS array (see Section 6.2.2). Risk factor curves are used to construct yield curves and other vector data types in a risk environment.

The following two steps are necessary in the construction of a yield curve (or another type of curve) in Risk Dimensions:

1. Register a risk factor variable for each one of a series of interest rates (or another type of rate), each with a different maturity value (see Section 5.2.4).
2. Create a risk factor curve from these risk factor variables.

Yield curves are generally used in the pricing functions of financial instruments like options and futures.

The following general form of the ***Array*** statement is used to declare risk factor curves in the *Proc Risk* procedure.

Array Name Role Elements = (   ) "Optional Attributes" ;

A suitable name for the risk factor array is specified in the ***name*** option. The names of the risk factor variables that are used in the risk factor curve are specified in the *Elements* option. Each of these variables must have the same value specified in the risk factor variable attribute *type*. There is no need to specify a ***type*** attribute for the risk factor curve as it has the same data type as the risk factor variables in the curve.

One of the following ***role*** attributes is specified for each risk factor curve:

- Var The elements of the array have an undefined or generic role.
- IR Interest rates are the elements of the array. This makes the declaration of the *Currency* attribute compulsory.
- FX The elements of the curve are exchange rates.

Optional attributes

The following optional attributes may be specified for the risk factor curves: *Category, Currency, Fromcur, Tocur, Group, Label* and *Refmap*. The descriptions of these attributes are the same as in Section 5.2.4.

Case study

Program Code 5.3 is used to create a risk factor curve named *Zero_Curve* in the *Casestudy_Env* risk environment.

Program Code 5.3: The registration of risk factor curves.

```
Proc Risk;
Environment open = "&RiskEnv";
Array Zero_Curve ir currency=ZAR
      Elements = (ZR_1_MTH  ZR_3_MTH ZR_6_MTH ZR_12_MTH  ZR_18_MTH
                  ZR_2_YEAR ZR_30_MTH ZR_3_YEAR ZR_42_MTH ZR_4_YEAR
                  ZR_54_MTH ZR_5_YEAR ZR_66_MTH ZR_6_YEAR ZR_78_MTH
                  ZR_7_YEAR ZR_90_MTH ZR_8_YEAR ZR_102_MTH ZR_9_YEAR
                  ZR_114_MTH ZR_10_YEAR)
      Refmap = (Zcurve = "ZAR");
/*The type attribute is omitted and the role attribute is set to
interest rate (ir). The Currency attribute is compulsory. The risk
factor variables that are listed in Elements statement, are included in
the risk factor curve. The Refmap attribute is discussed in more detail
in Section 5.2.7.*/
Environment save;
Run;
```

The yield curve, named *Zero_Curve* is used in the valuation of options, futures, government bonds and interest rate swaps in the *Casestudy_Env* risk environment. Consider an option. The Black Scholes pricing function (see Section 5.1) is used to calculate the value of an option. One input value in this pricing method is the risk-free rate of interest namely r that corresponds to the time to expiry (t). The yield curve named *Zero_Curve* is constructed from the risk factor variables *ZR_1_MTH*, *ZR_3_MTH*, …, *ZR_10_YEAR*. Each one of these risk factor variables has an interest rate value and a maturity value. Suppose the value of t falls between two maturity values on yield curve. The pricing function or method uses linear interpolation to calculate a value for r

from the interest rate values of these two neighbouring points. The use of the yield curve *Zero_Curve* is discussed in more detail in Section 7.3.

5.2.6 Output variables

A Risk Dimensions structure, called a method program is discussed in Chapter 7. One kind of method programs, called pricing methods is used to calculate the current value of each financial instrument in the portfolio. The calculated values are stored in the SAS data sets called **output data sets** that are created during the calculation process. **Output variables** are used to refer to data values that are derived from the calculated values in the data sets and the other data values of instrument variables. The output variables are also viewed in the output data sets.

An example is used to illustrate the concepts in the previous paragraph. In a pricing method the value of the instrument is computed and stored in the system-defined variable named *_VALUE_*. The instrument variable *Premium* contains the purchase price of the financial instrument. An output variable named *Daily_profit,* is assigned as *_VALUE_* minus *Premium*. The profit of keeping each instrument is viewed in the *Daily_Profit* variable or column of the output data set *Instvals*. This data set is created during the execution of the pricing method (see Section 10.6) and is stored in the *Output* library.

The general form of the ***Declare Outvars*** statement in *Proc Risk* follows. This statement is used to register output variables in risk environments.

```
Declare Outvars = (Name Type Role “Optional attributes”);
```

A valid SAS name is specified in the ***Name*** attribute.

The ***Type*** attribute has two available options, namely:

• Num	A numeric output variable is specified.
• Array[n]	An output variable that is an array of size n, is specified.

The ***Role*** attribute has to be specified as *computed*. A descriptive label may be declared in the *Label* attribute.

Several other optional attributes are also available for output variables. Examples are *Comptype*, *Postprice* and *Rollup*. The detail of these attributes is not discussed in this document.

Case study

Program Code 5.4 is used to create the output variable named *Daily_Profit* in the *Casestudy_Env* risk environment.

Program Code 5.4: The registration of output variables.

```
Proc Risk;
Environment open = "&RiskEnv";
/*The output variable (Role = computed) named Daily_Profit is of numeric
data type and has the optional attribute Label.*/
Declare Outvars = (Daily_Profit num computed
                   Label= "The MTM value minus Premium");
Environment save;
/*The Casestudy_Env is saved, with the added output variable declared
above*/
Run;
```

5.2.7 Reference variables

Reference variables are used to simplify the valuation process of financial instruments. Reference variables form part of a process called **reference mapping**. It enables the usage of a single pricing method for all the financial instruments of a certain type. Reference mapping is discussed at the end of this section.

Reference variables are declared by the ***Declare References*** statement in *Proc Risk*:

Declare References = (Name Type Length Role "Optional attributes");

One of the following data ***types*** is specified:

• Num	is a reference to a numeric Risk Dimensions variable.
• Array	is a reference to a risk factor array.
• Parameter	is a reference to a Risk Dimensions structure called a parameter matrix (see Section 9.2.2).

One of the following ***roles*** is specified:

• Var	is a reference to a generic risk factor variable.
• IR	is a reference to an interest rate variable.
• FX	is a reference to an exchange rate variable.
• FX_Spot	is a reference to a spot exchange rate.

The optional attributes *Label* and *Group* may also be specified for reference variables.

Program Code 5.5 is used to declare the necessary reference variables in the *Casestudy_Env* risk environment.

Program Code 5.5: The registration of reference variables.

```
Proc Risk;
/*The Casestudy_Env environment is opened*/
Environment open = "&RiskEnv";
Declare References = (Floatingrate num ir Label = "Ref to interest rate
                      in interest rate swap",
                      Price num var Label = "Ref to equity price",
                      Zcurve array var Label = "Ref to Zero curve");
Environment save;
/*The  Casestudy_Env  is  saved,  with  the  added  reference  variables
declared above*/
Run;
```

The explicit declaration of reference variables, as presented in Program Code 5.5 is not necessary. The reference variables were already assigned during the registration of risk factor variables and risk factor curves. The optional attribute with name *Refmap* was used during these registrations. Program Code 5.2 and 5.3 may be viewed to confirm this. The *Refmap* attribute is discussed in detail in the reference mapping process that is discussed in this section. If the reference variables are not registered separately, they are, however not listed under *Reference variables* option in the *Configuration* tree of the GUI.

Reference mapping

A process called **reference mapping** links open positions in the portfolio data to the appropriate market information in risk factor variables, to ensure the correct valuation of the instrument. This enables the usage of a single pricing method for all the instruments of a certain type.

Consider the following partial trade book and market information of the case study company *Activegrowth Limited*:

Trade book

Insttype	Instid	Short	Holding	Currency	**Underlying**	Enddate	Book	ContractPrice
FUTURE	SOL_Q42	0	12000	ZAR	**SOL**	17-Jun-04	Der	103.29
FUTURE	SOL_Q43	1	8000	ZAR	**SOL**	17-Jun-04	Der	99.55
FUTURE	AGL_Q41	0	5000	ZAR	**AGL**	17-Jun-04	Der	145.32
FUTURE	AGL_Q42	1	5000	ZAR	**AGL**	17-Jun-04	Der	156.43

Market information

Date	ASA	AGL	ISC	OML	SLM	SOL	JB_6_MTH
5/13/2004	45.00	133.50	32.90	11.50	8.55	99.00	0.07638

Valuation

The formula that is used to calculate the mark-to-market value of a future on an equity may be expressed in the following words:

Value = (Value-of-underlying-equity – Contractprice) x (some-discounting-factor)

The data values of the instrument attributes *Enddate* and *Contractprice* are necessary in the calculation of the formula. These values are obtained from the trade book or equivalently, the Risk Dimensions structure called a portfolio file (see Section 9.2.4). The value of the underlying equity, contained in the market data is also needed. The following **reference mapping steps** are necessary for a correct valuation of the future:

Step 1

The optional attribute *Refmap* is used in the reference mapping process to assign a reference variable and a reference key value to each risk factor variable that requires it. The assigning is done during the registration of the risk factor variable.

The generic form of the *Refmap* attribute is:

Refmap = (Refvar = “Refkey”)

The reference variable specified in *Refvar* is used in pricing methods. The reference key value specified in *“Refkey”* is used to find the name of a risk factor variable that refers to the market information that is used in pricing methods.

The following extract from the program code used in the case study illustrates the use of the *Refmap* attribute to assign reference variables:

Extract from Program Code 5.2: The registration of risk factor variables

```
Declare Riskfactors
(SOL       num  var  Group = "Commodity" Label = "Sasol Equity Price"
           Refmap = (Price = "SOL") Category = "Commodity"
           Mlevel = interval,
 AGL       num  var  Group = "Commodity" Label= "Anglo Equity Price"
           Refmap = (Price = "AGL") Category = "Commodity"
           Mlevel = interval );
```

The reference variable named *Price* and reference key values named *SOL* and *AGL* are assigned respectively.

Step 2

The second step of the reference mapping process is done in pricing methods (see Section 7.4.3). The correct references are made within the pricing method to enable the use of only one pricing function for instruments of the same type. The name of an instrument attribute is used to reference it. The reference to a risk factor variable has the following generic form:

Referencevariable.InstrumentAttribute

The following program code in the pricing method is used to value the future:

```
_VALUE_ = (Price.Underlying - ContractPrice)*discountingfactor);
```

The name of the reference variable is *Price* and the name of the instrument attribute is *Underlying*. The data values of *Underlying* provide a link to the risk factor variables *SOL* and *AGL*. The data values of these risk factor variables are necessary for correct valuation. It also follows that the values specified in the *Refkey* attributes in step 1 are data values of the instrument attribute *Underlying* in the trade book.

It is important to note that only step 1 of the reference mapping process takes place in this chapter. The second step namely the creation of pricing methods is discussed in Section 7.4.3. The pricing methods and the reference mapping process are only executed in Chapter 10. It is important in this chapter to create the correct reference variables and *Refmap* attributes. A closing remark is to note that reference variables are only used in pricing methods and not in other structures in Risk Dimensions.

5.2.8 Lag time grids

The pricing method of most financial instruments requires only position information and the current values of the risk factor variables. A **historical or lagged value** of a risk factor variable is, however necessary in the valuation of interest rate swaps. The value of the next floating payment is necessary in the valuation of the instrument. This value is usually determined by the value of the floating interest rate six months before the next exchange date. The historical value of the floating rate or corresponding risk factor variable has to be obtained for use in the pricing method.

The Risk Dimensions variable, named a **lag time grid** provides part of a solution to this problem. A lag time grid is defined as a list of **time element specifications** that is used to obtain lagged or historical values of risk factor variables. The historical values of the risk factor variables are stored in a **SAS data set**. The pricing method as discussed in Section 7.4.3 uses lag time grids to access the correct historical value in the SAS data set. Lagged values may be actual historical values, simulated values or linearly interpolated values and are defined with respect to the date of valuation.

The ***Timegrid*** statement in the *Proc Risk* procedure is used to define a lag time grid. A list of time element specifications is also assigned in this statement. Program Code 5.6 is used to create a lag time grid in the *Casestudy_Env* risk environment.

Program Code 5.6: The creation of a lag time grid in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
/*A lag time grid named Grid1 is created in the following statement. A
list of time element specifications is also assigned*/
Timegrid Grid1 (1 month, 45 days, 2 month, 75 days, 3 month, 4 month,
          5 month,6 month, 12 month);
Environment save;
Run;
```

In order to link a risk factor variable to the appropriate lag time grid, the *Laggrid* attribute is specified during risk factor variable registration. In the case study the risk factor variable *JB_6_MTH* is used to contain historical values of floating interest rates. The following extract from Program Code 5.2 illustrates how the risk factor variable *JB_6_MTH* is linked with the lag time grid named *Grid1.*

Extract from Program Code 5.2: The registration of risk factor variables

```
Declare Riskfactors =
(JB_6_MTH  num ir Label = "Floating Rate of Swap" Laggrid = grid1
           Refmap = (floatingrate = "JB_6_MTH") Currency = ZAR
           Maturity = 0.5 year Category = "IR",
```

It is important to note that the lag time grid named *Grid1* has to be created first, before it may be referred to in the *Laggrid* attribute. Thus, Program Code 5.6 has to be submitted before Program Code 5.2.

The lagged values of the risk factor variables are used in the pricing methods of instruments like interest rate swaps. There are two methods available to access these values, namely the use of **variable suffixes** or the ***Rlag()*** function.

Variable suffixes are used to retrieve the **actual** lagged values for all the available time points. The three suffixes available are:

• .lgrid	It returns the actual values of lag time grid points stated in terms of portion of a year.
• .lags	It returns the values of the risk factor variable at the points of the grid in correspondence with *.lgrid*.
• .nlag	It returns the size of the lag time grid.

The suffixes are concatenated with the name of the lag time grid when used in pricing methods, for example *Grid1.lgrid*, *Grid1.lags* and *Grid1.nlag*. The disadvantage of this method is that a lot of time points are not defined and a lot of lagged values are thus irretrievable. This method is not used for the case study.

The ***Rlag()*** function, used in pricing methods, makes use of **linear interpolation** to retrieve the lagged values. The general form of the function is:

Lagged_value = Rlag(Risk-factor-variable-name, Time-value);

The value specified in *Time-Value* is stated in terms of a portion of a year. The function does not make allowance for extrapolation. If the *Time-Value* specified is larger than the largest time element specified in the lag time grid, a missing value is returned. It is also important to note that the accuracy of the linear interpolation increases if more time points are specified in the lag time grid.

The following statement is used in the pricing method of interest rate swaps in the case study:

```
Flotrate = Rlag(floatingrate.ftr_name, (_date_ -fromdate)/365.25);
```

The use of lag time grids in pricing methods is discussed in more detail in Section 7.4.3.

5.2.9 The use of the GUI to view changes in the risk environment

The graphical user interface (GUI) is activated to view the changes that have been made by the *Proc Risk* procedure to the risk environment. The variables that were registered in the *Casestudy_Env* risk environment are used to illustrate the process of activating the GUI. This activation, may however be done after each execution of a *Proc Risk* procedure.

To activate the GUI, the word *risk* is typed into the *GUI* icon and the ✓ button is clicked.

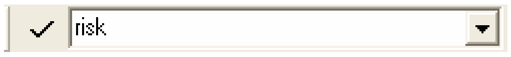
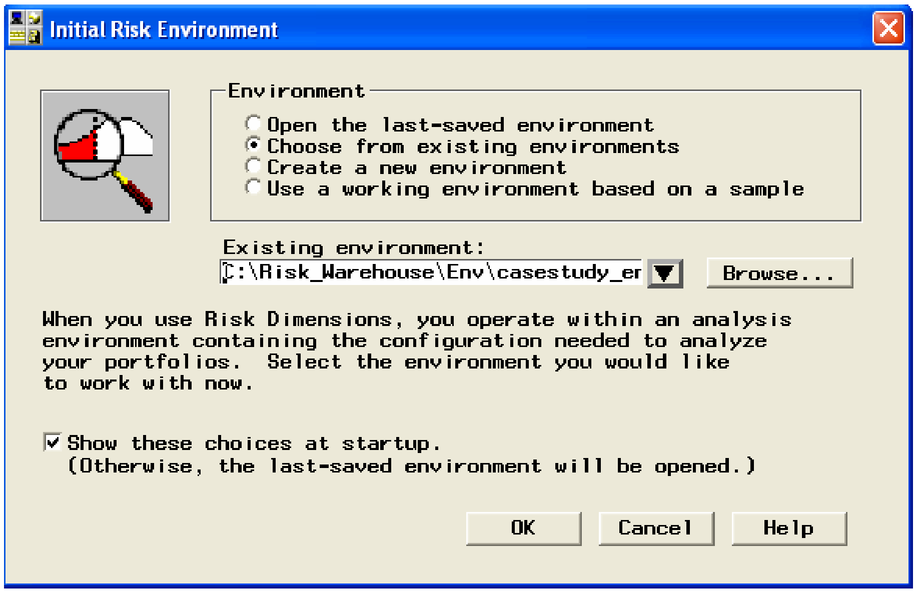


Figure 5.3: The GUI Icon

The *Initial Risk Environment Window* opens. The *Choose from existing environments* option is chosen. The *Browse* button is used to set the location and file name of the existing environment to *C:\Risk_Warehouse\Env\casestudy_env*. After these options have been chosen the *OK* button is clicked.



Open the last-saved environment
[✓ Choose from existing environments]
Create a new environment
Use a working environment based on a sample

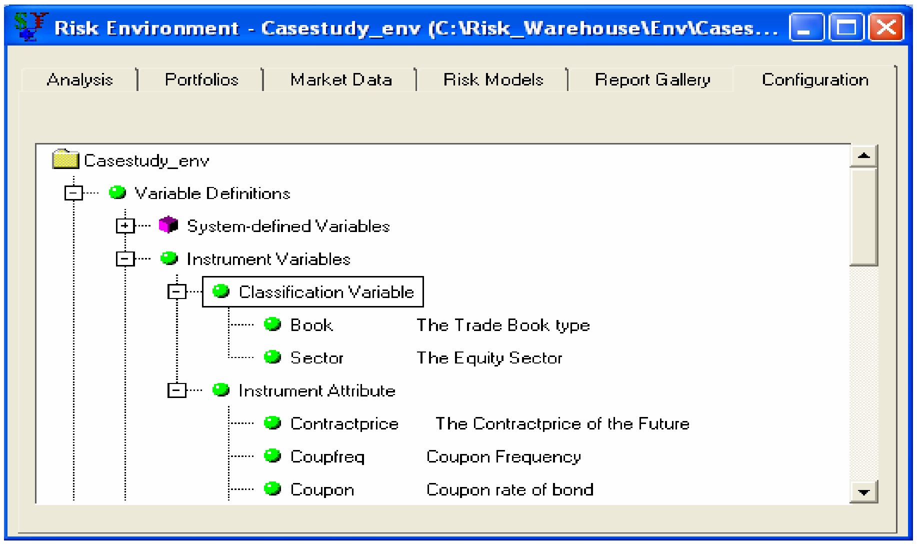
Existing environment:
C:\Risk_Warehouse\Env\casestudy_er

When you use Risk Dimensions, you operate within an analysis environment containing the configuration needed to analyze your portfolios. Select the environment you would like to work with now.

✓ Show these choices at startup.
(Otherwise, the last-saved environment will be opened.)

Figure 5.4: *The Initial Risk Environment Window*

The GUI opens and the contents of the *Casestudy_Env* risk environment are viewed. The variables that were created by Program Code 5.1 to 5.6 are viewed under the *Variable Definitions* option in the *Configuration* tree. The registered instrument variables are viewed in Figure 5.5 and are grouped according to the attribute *role*. A window that contains the attribute information about a variable, is opened when a variable name is double clicked on.



Casestudy_env
Variable Definitions
System-defined Variables
Instrument Variables
Classification Variable

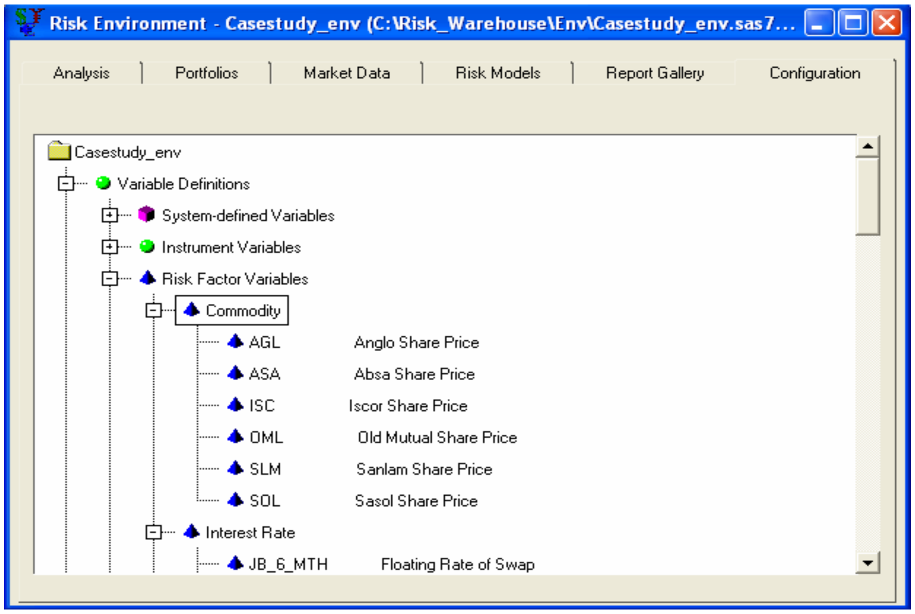
Book	The Trade Book type
Sector	The Equity Sector

Instrument Attribute

Contractprice	The Contractprice of the Future
Coupfreq	Coupon Frequency
Coupon	Coupon rate of bond

Figure 5.5: *The Variable Definitions option in the Configuration Tab*

The registered risk factor variables are viewed in Figure 5.6 and are grouped according to the *Group* attribute. A window containing the variable attribute information is opened when a variable name is double clicked on.



AGL	Anglo Share Price
ASA	Absa Share Price
ISC	Iscor Share Price
OML	Old Mutual Share Price
SLM	Sanlam Share Price
SOL	Sasol Share Price

JB_6_MTH	Floating Rate of Swap

Figure 5.6: *The Risk Factor variables in the Variable Definitions option*

The other kinds of variables may be viewed in a similar way.

5.3 Summary

The registration of Risk Dimensions variables was discussed in this chapter. It forms a very important part of the risk management system. All the necessary variables must be determined and registered in the correct way. The variables are used in the Risk Dimensions structures that are discussed in later chapters of this document.

6

DATA PREPARATION AND DATA-DRIVEN REGISTRATION

6.1 Introduction

The data values in SAS data sets are used in the execution of various risk analyses in Risk Dimensions. Different risk analyses require different data values and SAS data sets. The creation of the appropriate SAS data sets may be done by two methods. Table 6.1 contains a brief summary of the two methods.

Table 6.1: The data preparation methods

Steps	Method 1	Method 2
1	The creation of raw data files	The creation of raw data files
2	The modification and combination of raw data files	The conversion of raw data files into SAS data sets
3	The conversion of raw data files into SAS data sets	The modification and combination of SAS data sets

Consider the **first method**. The observed data values are captured in raw data files. The raw data files are then modified or combined to create new raw data files that contain the appropriate information for the risk analyses. These data files are created outside of the SAS window environment in a program such as

Microsoft Excel for example. The raw data files are converted or imported into SAS data sets. The data values in these sets are used in the execution of various risk analyses. The detail of the first method was discussed in Chapter 3.

Consider the **second method**. The observed market, position and other information, are again, captured in raw data files. These files are imported or converted into SAS data sets. These SAS data sets are modified and combined to form new SAS data sets that contain the appropriate data values for the risk analyses. Steps 1 and 2 of this method were also discussed in Chapter 3. Step 3 namely the **modification** and **combination** of SAS data sets are discussed in Sections 6.2 and 6.3 of this chapter.

A combination of the two methods is used in the case study. Data preparation is done inside and outside of the SAS window environment. Some preparation steps are easier in Microsoft Excel than in the SAS environment and vice versa.

An **alternative variable registration method**, namely **data-driven registration** may be used to register Risk Dimensions variables. The column names of SAS data sets, for example *Tradebook* and M*arket_History* are registered as instrument variables and risk factor variables. In order to understand and execute the process of data-driven registration, it is necessary understand the method of modifying SAS data sets.

6.2 The modification of SAS data sets

The SAS concepts and SAS structures that are used to modify SAS data sets are discussed in this section.

6.2.1 The basic Data step and column specification

The basic Data step is used to modify an existing SAS data set in this section. An input SAS data set is modified and stored as a SAS data set with either the same name or a new name. If the same name is specified, the new SAS data set replaces the existing SAS data set. If a new name is specified, a new SAS data set is created and the existing SAS data set still exists.

The Data step that is used is slightly different from the Data step that was discussed in Section 3.2.4. The new SAS data set that is created is specified in the *Data* statement. The *Set* statement is used to read the input SAS data set with all its observations (rows) and variables (columns). In order to overwrite an existing SAS data set, the same name is specified in both the *Data* and *Set* statements.

Not all the variables in the existing SAS data set need to be written to the new SAS data set. Two data set options, namely the *Drop* and *Keep* options are used to specify the variables that are written to the new SAS data set. The *Drop* option specifies the variables that are **not written** to the new SAS data set. The *Keep* option specifies the variables that **are written** to the new SAS data set. They are mutually exclusive options and only one is used, depending on the easiest to implement. The *Rename* option is used to give columns in the input SAS data set, new names in the new SAS data set. Example 6.1 is used to illustrate the basic modification of SAS data sets.

<u>Example 6.1: The specification of columns or variables in SAS data sets</u>

Consider the *Tradebook* SAS data set again. It has 49 observations and 14 variables. The variables are *Insttype, Instid, Shortposition, Holding, Currency, Premium, Underlying, Sector, Strikeprice, Enddate, Opttype, Book, ContractPrice and Marketprice*. The following data step is used to create a new SAS data set that contains all the observations, but only the following columns: *InstType, InstrumentId, Shortposition, Holding, Currency and Premium*. The variable *InstrumentId* is the new name for the variable *Instid* that was specified in the input data set.

Program Code 6.1: The registration of output variables

```
/*A new SAS data set named NewTradebook is created in the library
RiskData.  The Keep option specifies the variables that are included.*/
Data RiskData.NewTradebook (Keep = InstType InstrumentId
                            Shortposition Holding Currency Premium);
/*The input data set is Tradebook and is stored in the Riskdata library.
The variable Instid in the input SAS data set has the new name
InstrumentID in the new SAS data set*/
Set RiskData.Tradebook (rename = (Instid = InstrumentID));
Run;
```

The *Keep* option in Program Code 6.1 may be replaced by the following *Drop* option and will lead to the same result.

```
(Drop = Underlying Sector Strikeprice Enddate Opttype Book
        ContractPrice Shareprice)
```

6.2.2 Creating new variables

The **assignment** statement in the Data step is used to create **new variables** in existing SAS data sets. The general form of the assignment statement is:

New-Variable = expression;

The name of the new variable is specified in the *New-Variable* option. The ***expression*** is any valid combination of **constants**, **existing variables**, **operators**, **internal SAS functions** (see Table 6.3) and **parentheses**. Constants are values that are of character or numeric data type, for example the word *JIBAR* or the number *10*. All the variables in the input data set may be used in the expression. The use of SAS functions in the expression is discussed after

Example 6.2. Table 6.2 contains all the operators and parentheses that are valid for use in the expression of the assignment statement.

Table 6.2: Operators and parentheses

Symbol	Definition
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
( )	Grouping

Parentheses are used in program code to present expressions in a neat and easy understandable way. Parentheses may be used nested and each left parenthesis must have a corresponding right parenthesis.

The ***Length*** statement is used to define the length of data values of a variable in the SAS data set explicitly. The statement is made before the variable is used in an expression. If no length statement is used, the length of the data values of the variable is determined in the assignment statement. The general form of the length statement is:

```
Length variable(s) $ length;
```

Example 6.2 is used to illustrate the addition of new variables in an existing SAS data set. A new SAS data set is formed.

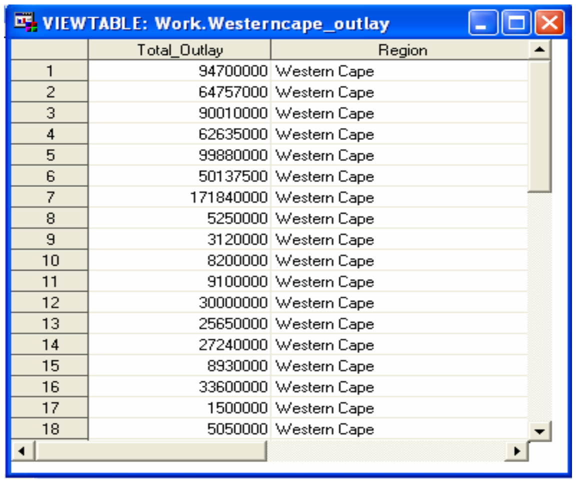
Example 6.2: The assignment of new variables and length specifications

Suppose new variables, for example *Total_Outlay* of numeric data type and *Region* of character data type are created for an existing SAS data set. The length statement is used to explicitly define the length of the data values of these variables. Program Code 6.2 is used to create these new variables and length specifications:

Program Code 6.2: The creation of new variables and length specifications

```
/*A new SAS data set named WesternCape_Outlay is created in the library
Work.  The Keep option specifies the variables that are included in the
new SAS data set.*/
Data Work.WesternCape_Outlay(keep=Total_Outlay Region);
/*The input data set Tradebook is stored in the Riskdata library.*/
Set RiskData.Tradebook;
/*The Length statement specifies the maximum length of the data values
for the new variables Total_Outlay and Region*/
Length Total_Outlay 8 Region $ 15;
/*The general form of the assignment statement is New-Variable =
expression;*/
Total_Outlay = Holding*Premium;
Region= "Western Cape";
Run;
```

The total outlay is the amount of units bought or sold (*Holding*) multiplied by the purchase price *(Premium)* per unit. The variable *Region* indicates the region where the units have been bought. The newly created SAS data set *WesternCape_Outlay* is viewed in Figure 6.1.



	Total_Outlay	Region
1	94700000	Western Cape
2	64757000	Western Cape
3	90010000	Western Cape
4	62635000	Western Cape
5	99880000	Western Cape
6	50137500	Western Cape
7	171840000	Western Cape
8	5250000	Western Cape
9	3120000	Western Cape
10	8200000	Western Cape
11	9100000	Western Cape
12	30000000	Western Cape
13	25650000	Western Cape
14	27240000	Western Cape
15	8930000	Western Cape
16	33600000	Western Cape
17	1500000	Western Cape
18	5050000	Western Cape

Figure 6.1: The *WesternCape_outlay* SAS data set

Internal SAS functions

A wide variety of built-in or internal **SAS functions** exist and are used to assign data values to new variables in the data step. The SAS functions are used in the *expression* option of the assignment statement. The SAS functions are divided into the following three groups: **SAS date functions**, **SAS character functions** and **SAS arithmetic functions**.

SAS date functions are used only in the Data step. Their input arguments are single data values or variables referring to many data values. The data type of the input arguments has to be in the correct format as illustrated below. The following SAS date functions are frequently used in data steps:

Year(SAS-date)

Calculates the appropriate year from a SAS date value and returns the year as a four digit value.

Qtr(SAS-date-value)

Calculates the appropriate quarter of year from a SAS date value and returns a number from 1 to 4.

Month(SAS-date-value)

Calculates the appropriate month from a SAS date value and returns a number from 1 to 12.

Day(SAS-date-value)

Calculates the appropriate day from a SAS date value and returns a number between 1 and 31.

Weekday(SAS-date-value)

Calculates the appropriate day of the week from a SAS date value and returns a number from 1 to 7, where 1 represents Sunday, 2 Monday etc.

Intnx("interval",start-from-date-value,increment)

A SAS date value is incremented by a specified number of intervals to form a new SAS date value. The original SAS date value is specified in the *start-from-date-value* option, the length of the intervals in *"interval"* and the number of increments in *increment*. The function returns the new SAS date value. For example, an *increment* of 1 and an *"interval"* of *"month"* advances the *start-from-date-value* forward to the date of the first day of the next month. If the *increment* was specified as *0*, the SAS date value of the first day of the month in which the *start-from-date-value* is contained would have been returned.

Intck("interval",date1,date2)

An interval length is specified in "*interval".* The function returns the number of these intervals between the two specified dates.

Mdy(month,day,year)

This function calculates the SAS date value from the specified *month*, *day* and *year* input values.

Today()

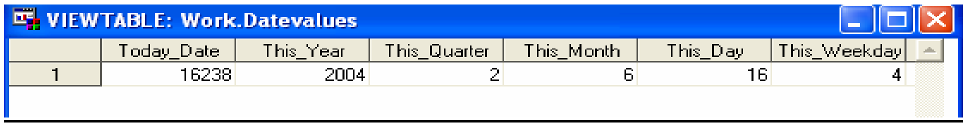
The data value of today is obtained from the system clock and returned in SAS date value format.

More SAS date functions exist, but are not discussed. Program Code 6.3 is used to illustrate the way in which SAS date functions are used in the data step.

Program Code 6.3: The use of SAS date functions

```
/*A SAS data set named DateValues is created in the temporary library
Work*/
Data Work.DateValues;
/*The Today() SAS function obtains the SAS date value of today from the
 internal clock (16238) and assigns it to variable Today_Date */
Today_Date = Today();
/*The year in which the variable Today_Date falls (2004) is extracted
 by the SAS function Year and is assigned to the variable This_Year*/
This_Year = Year(Today_Date);
/*The quarter of the year in which the variable Today_Date falls (2)
 is extracted by the SAS function Qtr and is assigned to the variable
This_Quarter*/
This_Quarter = Qtr(Today_Date);
/*The month in which the variable Today_Date falls (6) is extracted by
the SAS function Month and is assigned to the variable This_Month*/
This_Month = Month(Today_Date);
/*The day of the month on which the variable Today_Date falls (16) is
extracted by the SAS function Day and is assigned to the variable
 This_Day*/
This_Day= Day(Today_Date);
/*The day of the week on which the variable Today_Date falls (4) is
extracted by the SAS function Weekday and is assigned to the variable
This_Weekday*/
This_Weekday = Weekday(Today_Date);
/*The SAS function Intnx assigns the variable First_of_This_Month the
value of the first day of the month in which the variable Today_Date
falls (16223)*/
First_of_This_Month = Intnx("month",Today_Date,0);
/*The SAS function Intnx assigns the variable First_of_Next_Month the
value of the first day of the next month after the month in which
the variable Today_Date falls (16253)*/
First_of_Next_Month = Intnx("month",Today_Date,1);
/*The SAS function Intck counts the number of intervals (Days) between
the data values of the First_of_This_Month and First_of_Next_Month
variables. (31)*/
Number_Days = Intck("day",First_Of_This_Month,First_Of_Next_Month);
/*The SAS function Mdy assigns the SAS date value, obtained by the
input parameters contained in the following variables: This_Month,
This_Day and This_Year to the variable Todays_Date_Value (16238)*/
Todays_Date_Value = Mdy(This_Month,This_Day,This_Year);
Run;
```

The SAS data set named *Datevalues* in the *Work* library is viewed in Figure 6.2.


VIEWTABLE: Work.Datevalues

	Today_Date	This_Year	This_Quarter	This_Month	This_Day	This_Weekday
1	16238	2004	2	6	16	4

Figure 6.2: The *Datevalues* SAS data set

Although the use of the SAS date values is described for only one observation of the variables in Example 5.4, the program code is capable of handling variables with many data values.

SAS character functions are used to modify the data values of variables that are of character data type. New character variables are also created.

The following SAS character functions are used in Data steps:

Upcase(argument)

> All the character values in the argument are converted into the uppercase.

Scan(*argument,n,delimiter*)

> The *delimiter* specified splits the argument into parts. The *n* input parameter specifies which part of argument is returned by the function.

Index(*argument,extract*)

> The function returns the position of the first occurrence of the *extract* in the *argument.*

Substr(argument,position,n);

> The function is used to either extract or replace a substring of the *argument*. The placement in the expression determines the result of using this function. This concept is discussed in more detail in the example below.

Compress(argument, “characters- to-remove”)

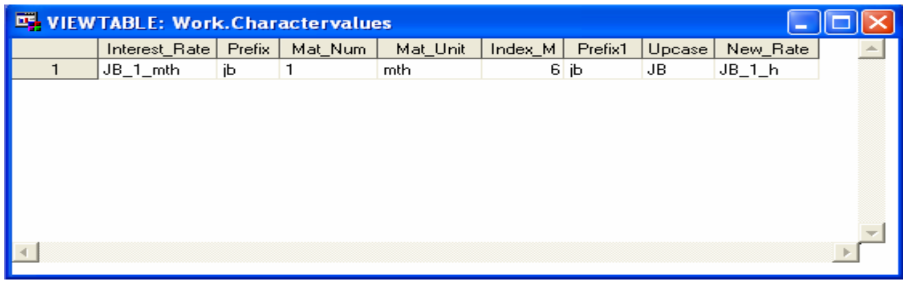
> This function removes specific characters from a character string. The specific characters are specified in the *“characters to remove”* option.

More SAS character functions exist, but are not discussed in this document. It is important to note that all character comparisons are case-sensitive. Program Code 6.4 is used to illustrate the use of SAS character functions in the data step.

Program Code 6.4: The use of SAS character functions

```
/*A data set named CharacterValues is created in the temporary library
Work*/
Data Work.CharacterValues;
/*The character string "jb_1_mth" is assigned as data value to the
variable Interest_Rate*/
Interest_Rate = "jb_1_mth";
/*The SAS function Scan is used to assign the character string  before
the first "_" of the data value of variable Interest_Rate (jb) to the
variable Prefix*/
Prefix = Scan(Interest_Rate,1,"_");
/*The SAS function Scan is used to assign the character string before
the second "_" and after the first "_" of the data value of variable
Interest_Rate(1) to the variable Mat_Num.*/
Mat_Num = Scan(Interest_Rate,2,"_");
/*The SAS function Scan is used to assign the character string before
the third "_" and after the second "_" of the data value of variable
Interest_Rate (mth) to the variable Mat_Unit.*/
Mat_Unit = Scan(Interest_Rate,3,"_");
/*The SAS function Index returns the position (6) of the extract "m" in
the data value of the variable Interest_Rate*/
Index_M = Index(Interest_Rate,"m");
/*The SAS function Substr extracts the first to second characters of the
data value of Interest_Rate (jb) and assigns this substring to the
variable Prefix1*/
Prefix1 = Substr(Interest_Rate,1,2);
/*The SAS function Upcase converts all the data values of the variable
Prefix1 and assigns this character string (JB) to the variable
Upcase_Prefix*/
Upcase_Prefix = Upcase(Prefix1);
/*The SAS function Substr replaces the first and second characters of
the data values of the variable Interest_Rate (jb) with the data values
contained in the variable Upcase_Prefix (JB)*/
Substr(Interest_Rate,1,2) = Upcase_Prefix;
/*The SAS function Compress removes the substring "mt" from the data
values of the variable Interest_Rate and assigns the remaining substring
(JB_1_h) to the variable New_Rate*/
New_Rate = Compress(Interest_Rate,"mt");
Run;
```

The SAS data set named *Charactervalues* that is grouped in the *Work* library is viewed in Figure 6.3.



	Interest_Rate	Prefix	Mat_Num	Mat_Unit	Index_M	Prefix1	Upcase	New_Rate
1	JB_1_mth	jb	1	mth	6	jb	JB	JB_1_h

Figure 6.3: The *Charactervalues* SAS data set

SAS arithmetic functions are used to create summary statistics in the data step. The following arithmetic functions are used:

Sum(argument,argument,….)

The function returns the sum of the input arguments.

Mean(argument,argument,….)

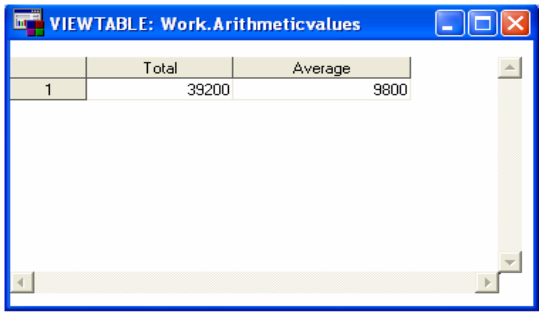
The function returns the arithmetic average of the input arguments.

Program Code 6.5 is used to illustrate the use of SAS arithmetic functions in the data step.

Program Code 6.5: The use of SAS arithmetic functions

```
/*A data set with the name ArithmeticValues is created in the temporary
library Work*/
Data Work.ArithmeticValues;
/*The SAS function Sum assigns the sum of the input values to the
variable Total*/
Total = Sum(10000,5000,12000,12200);
/*The SAS function Mean assigns the arithmetic mean of the input
parameters to the variable Average*/
Average = Mean(10000,5000,12000,12200);
Run;
```

The *Arithmeticvalues* SAS data set in the *Work* library is viewed in Figure 6.4.



	Total	Average
1	39200	9800

Figure 6.4: The *Arithmeticvalues* SAS data set

Conditional logic

Conditional statements like the *Else If-Then* statement are also used to create new or additional variables in the SAS data set. The value of a new variable is based on whether a specified condition in the *expression* statement is true or false.

The general form of the *Else If-Then* statement in the data step follows:

If *expression* ***then*** *statement;*
Else If *expression* ***then*** *statement;*
Else If *expression* ***then*** *statement;*
….

Table 6.2 contains the valid comparison operators that may be used in the *expression* of the *Else If-Then* statement.

Table 6.2: Comparison operators

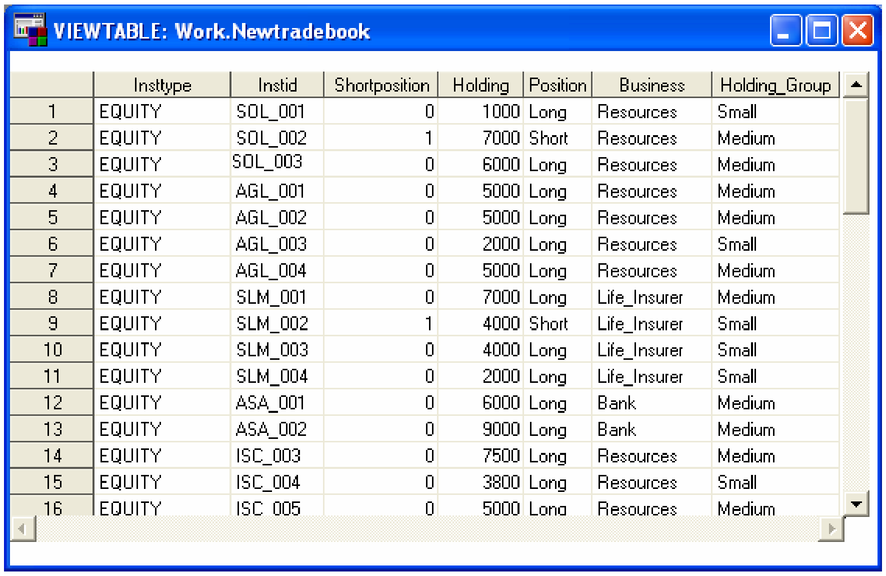
Letters	Symbol	Definition
EQ	=	equal to
NE	^=	not equal to
GT	>	greater than
LT	<	less than
GE	>=	greater than or equal to
LE	<=	less than or equal to
IN		equal to one of a list

Program Code 6.6 illustrates the use of conditional logic to create new variables to existing SAS data sets. The data values of the new variables *Position* and *Holding_Group* and *Business* are based on the data values contained in *Shortpostion*, *Holding* and *Instid* respectively.

Program Code 6.6: The use of conditional logic to create new variables in SAS data sets

```
/*The SAS data set NewTradebook is created in the temporary library
Work.  The Keep option specifies the variables that are included in this
data set.*/
Data Work.NewTradebook(keep = InstType Instid ShortPosition Position
                         Holding Holding_Group Business);
/*The Set statement specifies that the input SAS data set named
Tradebook is stored in the Riskdata library*/
Set Riskdata.Tradebook;
/*The following statement specifies the maximum length of the data
values of the new variables*/
Length Position $ 6
       Business $ 15
       Holding_Group $ 8;
/*The Else-If-Then statement and comparison operator "=" are used to
assign data values to the new variable Position*/
If ShortPosition = 1 then Position= "Short";
Else if ShortPosition = 0 then Position = "Long";
/*The Else If-Then statement, comparison operators "=", ">=", "<" and
the logical operator (and) are used to assign data values to the new
variable Holding_Group*/
If Holding < 5000 then Holding_Group = "Small";
Else if Holding >= 5000 and Holding < 10000 then
        Holding_Group = "Medium";
Else if Holding >= 10000 then Holding_Group = "Large";
/*The Else-If-Then statement, the Substr function and comparison
operator (in) are used to assign data values to the new variable
Business*/
If
Substr(Instid,1,3) in ("SOL" "AGL" "ISC") then Business = "Resources";
Else if
Substr(Instid,1,3) in ("OML" "SLM") then Business = Life_Insurer";
Else if
Substr(Instid,1,3) = 'ASA' then Business = "Bank";
Run;
```

The SAS data set named *Newtradebook* in the *Work* library is viewed in Figure 6.5.



	Insttype	Instid	Shortposition	Holding	Position	Business	Holding_Group
1	EQUITY	SOL_001	0	1000	Long	Resources	Small
2	EQUITY	SOL_002	1	7000	Short	Resources	Medium
3	EQUITY	SOL_003	0	6000	Long	Resources	Medium
4	EQUITY	AGL_001	0	5000	Long	Resources	Medium
5	EQUITY	AGL_002	0	5000	Long	Resources	Medium
6	EQUITY	AGL_003	0	2000	Long	Resources	Small
7	EQUITY	AGL_004	0	5000	Long	Resources	Medium
8	EQUITY	SLM_001	0	7000	Long	Life_Insurer	Medium
9	EQUITY	SLM_002	1	4000	Short	Life_Insurer	Small
10	EQUITY	SLM_003	0	4000	Long	Life_Insurer	Small
11	EQUITY	SLM_004	0	2000	Long	Life_Insurer	Small
12	EQUITY	ASA_001	0	6000	Long	Bank	Medium
13	EQUITY	ASA_002	0	9000	Long	Bank	Medium
14	EQUITY	ISC_003	0	7500	Long	Resources	Medium
15	EQUITY	ISC_004	0	3800	Long	Resources	Small
16	EQUITY	ISC_005	0	5000	Long	Resources	Medium

Figure 6.5: The *NewTradebook* SAS data set

The *Else If-Then* statement may also be used in method programs that are discussed in Section 7.3.

The do loop

The **do loop** is used to perform repetitive calculations and thus simplify program code. This loop is used in data steps or in method programs.

The general form of the **simple iterative do** loop is:

Do *index-variable* = *start* **To** *stop* **By** *increment*;
SAS statements
End;

The *start*, *stop* and *increment* values are established at the start of the Do loop and cannot be changed during the iterative process. The default value specified for *increment* is 1.

The structure of the do loop may be modified into a **do loop with a value list** that has the following general structure:

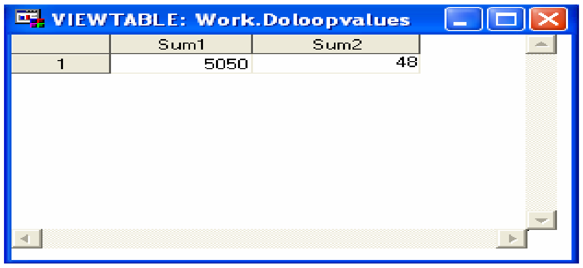
Do *index-variable* = *value1, value2, value3, ...;*
SAS statements
END;

Program Code 6.7 illustrates the use of both forms of the **do loop**.

Program Code 6.7: Both forms of the do loop

```
/*The SAS data set DoLoopValues is created in the temporary library
Work.  The Drop option specifies the variables that are not included in
the data set.*/
Data Work.DoLoopValues(drop = n);
/*The values of the variables, Sum1 and Sum2 are initialized as nought*/
Sum1=0;
Sum2=0;
/*The general form of the Do loop is used to calculate the sum of the
the numbers 1 to 100*/
Do n = 1 to 100;
Sum1 =Sum1 + n;
End;
/*The Do loop with a value list is used to calculate the sum of the
values specified in the value list*/
Do n= 1,3,5,7,8,9,15;
Sum2 = Sum2+n;
End;
Run;
```

The SAS data set *DoLoopvalues* in the *Work* library is viewed in Figure 6.6.



	Sum1	Sum2
1	5050	48

Figure 6.6: The *DoLoopvalues* SAS data set

SAS Arrays

In the SAS program language, a SAS array is not a data structure, but rather a temporary grouping of certain variables in a specified order. A SAS array is identified by an array name, is not a variable and exists only for the current duration of the data step that it is used in. Each value in an array is called an element and is referenced by a subscript that represents the position of the element in the array. SAS arrays are used in data steps, user-defined SAS functions (see Section 7.2) and method programs. They are also used with the do loop to perform repetitive calculations.

In Section 5.2.5 the Risk Dimensions variable, named risk factor curves or risk factor arrays were discussed. Risk factor arrays are a special type of a SAS array. In the following extract from Program Code 5.3 the use of the *Array* statement to create SAS arrays is illustrated. The name of the array is specified as *Zero_Curve* and the elements are specified as *ZR_1_MTH*, *ZR_3_MTH*,…, *ZR_10_YEAR*.

Extract from Program Code 5.3: The registration of risk factor arrays

```
Array Zero_Curve ir Currency = ZAR
      Elements = (ZR_1_MTH  ZR_3_MTH  ZR_6_MTH  ZR_12_MTH  ZR_18_MTH
                  ZR_2_YEAR ZR_30_MTH ZR_3_YEAR ZR_42_MTH  ZR_4_YEAR
                  ZR_54_MTH ZR_5_YEAR ZR_66_MTH ZR_6_YEAR  ZR_78_MTH
                  ZR_7_YEAR ZR_90_MTH ZR_8_YEAR ZR_102_MTH ZR_9_YEAR
                  ZR_114_MTH ZR_10_YEAR)
      Refmap =(Zcurve = "ZAR");
```

The do loop that was discussed earlier in this section is used together with SAS arrays to perform repetitive calculations. Suppose that the value of each element of *Zero_Curve* has to be incremented by one percentage point, then Program Code 6.8 is used to make this adjustment.

Program Code 6.8: The do loop and SAS arrays in repetitive calculations

```
/*Each element is referenced by its position in the array, for example
Zero_Curve{5} refers to ZR_18_MTH*/
Do i = 1 to 22;
Zero_Curve{i} = Zero_Curve{i}+ 0.01;
End;
```

The use of SAS arrays and do loops to perform repetitive calculations are further illustrated in Section 7.3.

6.2.3 Controlling the rows of a SAS data set

Consider the Data step again. The name of the new SAS data set is specified in the *Data* statement and the name of the input SAS data set is specified in the *Set* statement. Two new statements, namely the ***If*** statement and ***Where*** statement are used to **control the rows** of the new SAS data set. These statements select a subset of the rows of the input SAS data set and only these rows are written to the new SAS data set. The data values of the variables (columns) of the SAS data set are used in these statements to determine the subset of rows. The variables are, either already in the input SAS data set or are newly created variables in the Data step.

The use of the *If* and *Where* statements in terms of controlling the rows are illustrated in terms of the SAS data set with name *Tradebook* that is stored in the *Riskdata* library.

The ***If* statement** is used only in the data step and has the following general form:

```
If expression;
```

The following program code ensures that only rows that contain observations of *Equity* or *Future* as data values for the *InstType* variable are included in the new SAS data set. The other rows are excluded.

```
If InstType in ("Equity" "Future");
```

The ***Where* statement** is used in both Data and Proc steps and has the following general form:

```
Where expression;
```

In the following program code all the rows that **do not** have *ASA* as observation of the data value for the variable *Underlying* are included in the new SAS data set.

```
Where Underlying ^= "ASA";
```

The following **special operators** may also be used in the *Where* statement:

Like

> The operator selects observations by comparing character values to specified patterns. A percentage sign "%" replaces any number of characters and an underscore "_" replaces one character.

=*

> The operator selects observations that contain a spelling variation of the word or words that are specified.

Contains or *?*

> Only observations that include a specified substring are included in the new data set.

Is null or *is missing*

Observations that have missing values for the specified variable are included in the new data set.

Between-and

The operator selects observations in which the value of the variable falls within a range of values.

The use of each special operator that was discussed above is illustrated by Program Code 6.9 to Program Code 6.13.

Program Code 6.9: The special operator *like*

```
/*All the observations of the variable Instid that starts with a "S" are
included in the new SAS data set.*/
Where Instid like "S%";
```

Program Code 6.10: The special operator =*

```
/*All the observations that have data values for the variable Instid
that are "SLM" or a variation there of, are included in the new SAS data
set.*/
Where Instid =* "SLM";
```

Program Code 6.11: The special operator *Contains*

```
/*All the observations that have the substring "AGL" as part of their
data values for the variable Instid are included in the new SAS data
set*/
Where Instid Contains "AGL";
```

Program Code 6.12: The special operator *is missing*

```
/*All the observations that have a missing data value for the variable
OptType are included in the new SAS data set*/
Where OptType is missing;
```

Program Code 6.13: The special operator *between-and*

```
/*All the observations that have a data value for Holding that lies
between 3000 and 8000 are included in the new SAS data set*/
Where Holding between 3000 and 8000;
```

It is important to note that when a *Where* statement follows another *Where* statement in the Data or Proc step, it replaces the previous one.

The modification of SAS data sets in the context of the case study is discussed in Section 6.4.

6.3 The combination of SAS data sets

The methods that are available for the combination of two or more SAS data sets into a single SAS data set are discussed in this section. **Concatenation** and **interleaving** is discussed in Section 6.3.1. **Match-merging** is discussed in Section 6.3.2.

6.3.1 The Concatenation and Interleaving SAS data sets

A **concatenation** combines two or more SAS data sets, one after the other, into a single SAS data set. The *Set* statement in the Data step is used to perform the concatenation. The new data set contains all the observations (rows) of the original data sets in sequential order, as well as, all the variables (columns). The names of the variables of the original data sets may differ. If this happens the observations from the one data set have missing values for variables defined only in other data set.

The general form of the Data step for concatenation:

```
Data  SAS-data-set;
Set   Sas-data-set-1 SAS-data-set-2;
Run;
```

The order of the data sets specified in the *Set* statement determines the order of the observations in the new data set. Any number of data sets may be specified.

Interleaving is a more refined method of combining two or more data sets into a single data set. The original data sets are first sorted according to the data values of a specific variable, for example *InstType*. The *Set* statement is again used in the Data step to specify the names of the original data sets. An additional statement in the Data step, the *By* statement is used to specify the order of the observations in the new data set according to the data values of a variable for example *InstType*. The data sets that are created through interleaving contain all the observations and variables from the original data sets.

The SAS procedure named ***Proc Sort*** is used to sort the observations of the original data sets according to the data values of a specific variable.

The general form of *Proc Sort* is:

```
Proc Sort  Data = Input-SAS-data-set ;
           Out  =  Output-SAS-data-set ;
By  <Descending> variable-name ;
Run;
```

The name of the unsorted SAS data set is specified in the *Data* option. The name that is assigned to the resulting sorted SAS data set, is specified in the *Out* option. If this option is omitted the sorted version of the data set specified in the *Data* option replaces the original data set. The order of sorting is defined as descending by using the word *Descending* in the *By* statement. If no word is specified in this statement the order of sorting is ascending.

The procedure *Proc Sort* rearranges the observations in SAS data sets, can sort on multiple variables in descending or ascending order and treats missing values as the smallest possible values.

The Data step with a *Set* and *By* statement is used to interleave the sorted SAS data sets. The general form of the Data step that is used in interleaving follows:

Data *SAS-data-set*;
Set *SAS-data-set-1 SAS-data-set-2*;
By *<Descending> variable-name*;
Run;

The name of the new SAS data set, is specified in the *Data* statement and the names of the sorted input SAS data sets are included in the *Set* statement. The data values of the variable specified in *variable-name* are used to determine the order of the observations in the new data set. The order of the observations may be ascending or descending as specified in the *By* statement.

The concatenation and interleaving concepts are illustrated in Example 6.3.

Example 6.3: The concatenation and interleaving of SAS data sets

Consider SAS data sets, *Book1* and *Book2* in the temporary SAS library *Work*. The data sets have the following form:

Work.Book1:

Insttype	**Premium**	**Region**
Option	7.5	WCape
Future	0	WCape
Equity	42.4	WCape

Work.Book2:

Insttype	**Premium**
Gov_Bond	89.3
Int_Swap	0
Gov_Bond	90.2

Program Code 6.14 is used **concatenate** *Book1* and *Book2* into the SAS data set with name *FinalBook*.

Program Code 6.14: The concatenation of *Book1* and *Book2*

```
Data Work.Finalbook;
Set Work.Book1 Work.Book2;
Run;
```

The resulting structure of *Finalbook* follows:

Insttype	Premium	Region
Option	7.5	WCape
Future	0	WCape
Equity	42.4	WCape
Gov_Bond	89.3	
Int_Swap	0	
Gov_Bond	90.2	

The input SAS data sets were combined one after the other in the new SAS data set. The observations of the second input data set have missing values for the variable *Region* that are found only in the first input data set.

In Program Code 6.15 the SAS data sets *Book1* and *Book2* are used to illustrate the concept of **interleaving**.

Program Code 6.15: The interleaving of *Book1* and *Book2*

```
/*The sorted versions of the SAS data set Book1 and Book2 in the Work
library replace their unsorted versions. The observations of both data
sets are sorted in ascending order according to the data values of the
Insttype variable */
Proc Sort Data = Work.Book1;
By Insttype;
Run;
Proc Sort Data = Work.Book2;
By Insttype;
Run;
/*The rows of the new SAS data set Finalbook in the Work library are in
ascending order according to the data values in the Insttype variable*/
Data Work.Finalbook;
Set Book1 Book2;
By Insttype;
Run;
```

The resulting structure of *Finalbook* follows:

Insttype	Premium	Region
Equity	42.4	WCape
Future	0	WCape
Gov_Bond	89.3	
Gov_Bond	90.2	
Int_Swap	0	
Option	7.5	WCape

The observations of the new data set are in alphabetical or ascending order according to the data values of the *Insttype* variable.

6.3.2 Match-merging SAS data sets

Match-merging is another method of combining the observations of two or more SAS data sets into a single SAS data set. **One or more observations** from the original data sets are combined into a **single observation** in the final SAS data set according to the data values of a common variable. The number of observations in the new data set is less or equal to the sum of the observations of the original data sets.

If all the variables of the original data sets contributed to an observation in the new data set, a **match** occurs. If this is not the case, the observation is a **non-match**.

Consider the observations of the original data sets. The observations in these data sets may have a **one-to-one**, a **one-to-many** or a **many-to-many** relationship. If a single observation in one data set is related to a single observation in another data set then a one-to-one relationship holds. In a one-to-many relationship, unique observations in one data set are related to multiple observations in the second data set. If multiple observations in one data set are related to multiple observations in another data set, a many-to-many relationship holds.

The observations of the data sets that are used in the match-merging process are first sorted by *Proc Sort* (see Section 6.3.1) according to the values of a common variable. The Data step with a ***Merge*** and *By* statement are used for the match-merging process.

The general form of the step follows:

```
Data   SAS-data-set;
Merge   SAS-data-set-1   (IN=IN1)
          SAS-data-set-2   (IN=IN2);
By variable-name;
Run;
```

The *Data* statement specifies the name of the final SAS data set. The names of the input SAS data sets are specified in the *Merge* statement. The *In* options in the *Merge* statement are used in determining matches and non-matches. The variable specified in the *In* option (*In1* or *In2*) has a value of 1 (match) when the data set contributed to the observation in the final data set. In the case of a non-match, the variable has a value of 0. All the observations, matches and non-matches, except the variables *In1* and *In2*, are written by default to the final SAS data set. The *If* statement control may be used to write only matches or non-matches to the final data set.

If the same variable name occurs in more than one data set but the variables contain different information, the result of the match-merging may be unsatisfactory. The *Rename* data set option in the *Merge* statement is used to change the name of a variable from an input data set in the final data set. The general form of the *Rename* data set option follows:

```
SAS-data-set(Rename= (old-name = new-name));
```

The *Rename* option is also useful when the variable specified in the *By* statement has a different name in each of the original data sets. This option is used to change the names so that they match. The new variable name is then specified in the *By* statement. Example 6.4 is used to illustrate the concept of match-merging.

Example 6.4: The match-merging of SAS data sets

Consider SAS data sets, *Book* and *Traderinfo* in the temporary SAS library *Work*. Information about the open positions held is in *Book* variable. The identification number of the trader for the transaction is in the *TraderID* variable. The name of the instrument is stored in the *Insttype* variable and the date of the transaction in the variable *Date*. Further information about the traders that carry transactions out is stored in the *Traderinfo* SAS data set. The variable *TraderID* again refers to an identification number of the trader, the variable *Trader* to the name of the trader and the variable *Date* to the date that the trader was hired. The data sets have the following form:

Work.Book:

TraderID	**Insttype**	**Date**
11384	Equity	05/11/04
27604	Future	04/13/04
27604	Equity	05/04/04
11384	Option	05/05/04
35001	Future	05/11/04
27604	Option	05/10/04

Work.Traderinfo:

TraderID	**Trader**	**Date**
11384	Jacobs	03/04/93
35001	Smith	10/11/03
20030	Adams	07/03/96
27604	Barnard	09/09/00

The original data sets *Book* and *Traderinfo* are combined into a single data set, named *Finalset.* In both data sets *Book* and *Traderinfo* a variable *Date* exists, but these variables contain different information. The *Rename* option is used to solve this problem. Program Code 6.16 and 6.17 are used to match-merge the two data sets in a single data set, making allowance first for non-matches and then matches.

Program Code 6.16: Match merging with **non-matches**:

```
/*The SAS data sets Book and Traderinfo in the Work library are first
sorted.  The sorted versions of these SAS data sets replace their
unsorted versions. The observations of both data sets are sorted in
ascending order according to the data values of the TraderID variable */
Proc Sort Data = Work.Book;
By TraderID;
Run;
Proc Sort Data = Work.Traderinfo;
By TraderID;
Run;
/*The name of the combined SAS data set is Finalset and is specified in
Data statement. The Date variables in Book and Traderinfo are renamed as
TradeDate and DateOfHire in Finalset respectively.  The observations of
Finalset are created by the data values of the common variable
TraderID.*/
Data Work.Finalset;
Merge Work.Book (Rename = (Date = TradeDate))
      Work.Traderinfo (Rename = (Date = DateOfHire));
By TraderID;
Run;
```

The resulting SAS data set with name *Finalset* in the *Work* library follows:

TraderID	Insttype	TradeDate	Trader	DateOfHire
11384	Equity	05/11/04	Jacobs	03/04/93
11384	Option	05/05/04	Jacobs	03/04/93
20030			Adams	07/03/96
27604	Future	04/13/04	Barnard	09/09/00
27604	Equity	05/04/04	Barnard	09/09/00
27604	Option	05/10/04	Barnard	09/09/00
35001	Future	05/11/04	Smith	10/11/03

The third observation of the data set is a non-match observation. The trader with name *Adams* has no transactions in the trade book. No data values for variables of *Book*, namely *InstType* or *TradeDate* are associated with this observation.

Program Code 6.17 is used to match-merge the two data sets in a single data set, making only allowance for matches:

Program Code 6.17 Match-merging with **matches**:

```
/*Assume that the original SAS data sets have been sorted according to
the data values of the TraderID variable. The In data set option is used
to control matches.  The matching observations have a value of 1 for
both the temporary variables Inbook and InInfo. The If statement is used
to select the subset of matching observations. */
Data Finalset;
Merge Work.Book (Rename = (Date = TradeDate) In = InBook)
      Work.Traderinfo (Rename=(Date = DateOfHire)In = InInfo);
By TraderID;
If Inbook = 1 and InInfo = 1;
Run;
```

The structure of the SAS data set with name *Finalset* follows:

TraderID	Insttype	TradeDate	Trader	DateOfHire
11384	Equity	05/11/04	Jacobs	03/04/93
11384	Option	05/05/04	Jacobs	03/04/93
27604	Future	04/13/04	Barnard	09/09/00
27604	Equity	05/04/04	Barnard	09/09/00
27604	Option	05/10/04	Barnard	09/09/00
35001	Future	05/11/04	Smith	10/11/03

6.4 Case study: The modification and combination of SAS data sets

The concepts that were discussed in Sections 6.2 and 6.3 are illustrated in terms of the case study in this section. A statistical analysis method, named **principal components analysis** (see Section 6.4.1) is applied to the SAS data set, named *Yieldcurve_data*. The results of the analysis are stored in other SAS data sets. The information stored in these SAS data sets is combined with various other SAS data sets to form new SAS data sets. The **final SAS data sets** are used in Risk Dimensions. A theoretical discussion of **variance-covariance matrices** is included in Section 6.4.2. The variance-covariance matrix is stored in a SAS data set and is used in covariance-based Monte Carlo simulation. This simulation method is one of the methods that is used to calculate Value at Risk and is discussed in more detail in Chapter 10. The implementation of all these concepts, in terms of the case study, is discussed in Section 6.4.3.

6.4.1 Principal Components Analysis

The theoretical aspects of the statistical technique, named **principal components analysis** are discussed in this section.

The definition of **eigenvalues** and **eigenvectors** are made first (Johnson et al. (2002)).

Definition: Eigenvalues

Let A be a square matrix of size $k \times k$. Let I be the identity matrix of the same size $k \times k$. The scalars $\lambda_1, \lambda_2, \ldots, \lambda_k$ that satisfy the polynomial equation

$$\left| A - \lambda I \right| = 0 \tag{6.1}$$

are called the eigenvalues of a matrix A.

Definition: Eigenvectors

Let A be a square matrix of size $k \times k$ and let λ be an eigenvalue of A. If the non-zero vector $\mathbf{x}$ of dimension $k \times 1$ is such that

$$A\mathbf{x} = \lambda \mathbf{x} \tag{6.2}$$

then $\mathbf{x}$ is said to be an eigenvector of the matrix A associated with the eigenvalue λ.

Consider p random variables named $X_1, X_2, \ldots, X_p$. **Principal components** are defined as particular linear combinations of these variables. Suppose further that $X_1, X_2, \ldots, X_p$ have a covariance matrix Σ and a correlation matrix ρ. Principal component analysis may be executed in two different ways that leads to slightly different results. The eigenvalues and eigenvectors of either the covariance matrix Σ or the correlation matrix ρ are used in the analysis. The **covariance matrix method** is discussed first. The results and concepts applied in this method are also used in the correlation matrix method.

Consider the random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_p]$ with a covariance matrix Σ that has eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$. Consider further the following linear combinations:

$$\begin{aligned} Y_1 &= \mathbf{a_1}'\mathbf{X} = a_{11}X_1 + a_{12}X_2 + \ldots + a_{1p}X_p \\ Y_2 &= \mathbf{a_2}'\mathbf{X} = a_{21}X_1 + a_{22}X_2 + \ldots + a_{2p}X_p \\ &\vdots \\ Y_p &= \mathbf{a_p}'\mathbf{X} = a_{p1}X_1 + a_{p2}X_2 + \ldots + a_{pp}X_p \end{aligned}$$

where $\mathbf{a_i} = [a_{i1}, a_{i2}, \ldots, a_{ip}]$ denotes a vector of size $(p \times 1)$.

It follows that:

$$\begin{aligned} Var(Y_i) &= \mathbf{a_i}'\Sigma\mathbf{a_i} && \text{for } i = 1,2,\ldots,p \\ Cov(Y_i, Y_k) &= \mathbf{a_i}'\Sigma\mathbf{a_k} && \text{for } i,k = 1,2,\ldots,p \end{aligned}$$

The principal components are defined as those **uncorrelated linear combinations** $Y_1, Y_2, \ldots Y_p$ whose variances are as large as possible.

The first principal component is the linear combination with the maximum variance. Thus it is the combination that maximizes $Var(Y_1) = \mathbf{a_1}'\Sigma\mathbf{a_1}$. The values of the coefficient vectors are restricted to unit length.

It follows that:

The first principal component is the linear combination $\mathbf{a_1}'\mathbf{X}$ that maximizes $Var(\mathbf{a_1}'\mathbf{X})$ subject to $\mathbf{a_1}'\mathbf{a_1} = 1$

The second principal component is the linear combination $\mathbf{a}_2'\mathbf{X}$ that maximizes $Var(\mathbf{a}_2'\mathbf{X})$ subject to $\mathbf{a}_2'\mathbf{a}_2 = 1$ and
$Cov(\mathbf{a_1}'\mathbf{X}, \mathbf{a_2}'\mathbf{X}) = 0$

The i'th principal component is the linear combination $\mathbf{a}_i'\mathbf{X}$ that maximizes
$Var(\mathbf{a}_i'\mathbf{X})$ subject to $\mathbf{a}_i'\mathbf{a}_i = 1$ and
$Cov(\mathbf{a}_i'\mathbf{X}, \mathbf{a}_k'\mathbf{X}) = 0$ for $k < i$

Results 6.1 and 6.2 (Johnson et al. (2002)) contain essential results that are used in principal component analysis.

Result 6.1

Consider the random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_p]$ again with corresponding covariance matrix Σ. Further let Σ have the eigenvalue-eigenvector pairs

$$(\lambda_1, \mathbf{e_1}), (\lambda_2, \mathbf{e_2}), \ldots, (\lambda_p, \mathbf{e_p})$$

where

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p.$$

Then the i'th principal component is given by

$$Y_i = \mathbf{e_i}'\mathbf{X} = e_{i1}X_1 + e_{i2}X_2 + \ldots e_{ip}X_p \qquad \text{for} \qquad i = 1,2,\ldots,p \qquad (6.3)$$

It also follows that:

$$Var(Y_i) = \mathbf{e_i}'\Sigma\mathbf{e_i} = \lambda_i \text{ for } i = 1,2,\ldots,p \quad (6.4)$$

$$Cov(Y_i, Y_k) = \mathbf{e_i}'\Sigma\mathbf{e_k} = 0 \text{ for } i \neq k \quad (6.5)$$

If some of the eigenvalues λ_i are equal, then the corresponding eigenvectors e_i and thus Y_i are not unique. It thus follows that the coefficient vector that maximizes $Var(Y_1)$ is the eigenvector of the largest eigenvalue, namely λ_1.

Result 6.2

Consider the random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_p]$ with corresponding covariance matrix Σ. Further let Σ have the eigenvalue-eigenvector pairs

$$(\lambda_1, \mathbf{e}_1), (\lambda_2, \mathbf{e}_2), \ldots, (\lambda_p, \mathbf{e}_p)$$

where

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p.$$

Further let

$$Y_1 = \mathbf{e_1}'\mathbf{X}, Y_2 = \mathbf{e_2}'\mathbf{X}, \ldots, Y_p = \mathbf{e_p}'\mathbf{X}$$

denote the principal components.

It follows that:

$$\begin{aligned} &\textit{Total Variance} \\ &= \sigma_{11} + \sigma_{22} + \ldots + \sigma_{pp} \\ &= \sum_{i=1}^{p} Var(X_i) \\ &= trace(\Sigma) \\ &= \lambda_1 + \lambda_2 + \ldots + \lambda_p \\ &= \sum_{i=1}^{p} Var(Y_i) \end{aligned} \quad (6.6)$$

It also follows that that the proportion of the total population variance that is explained by the k'th principal component is equal to:

$$\frac{\lambda_k}{\lambda_1 + \lambda_2 + \ldots \lambda_p}.$$

The first few principal components usually contribute a large proportion of the total population variation (usually 80% to 90%). The original p factors may then be replaced by these few principal components without much loss of information.

The variables $\mathbf{X}' = [X_1, X_2, \ldots, X_p]$ are standardized in the **second method** of principal component analysis. Consider the following standardized variables:

$$Z_1 = \frac{(X_1 - \mu_1)}{\sqrt{\sigma_{11}}}$$

$$Z_2 = \frac{(X_2 - \mu_2)}{\sqrt{\sigma_{22}}}$$

$$\vdots$$

$$Z_p = \frac{(X_p - \mu_p)}{\sqrt{\sigma_{pp}}} \tag{6.7}$$

The standardized variables may also be written in matrix notation as

$$\mathbf{Z} = (V^{1/2})(\mathbf{X} - \boldsymbol{\mu}) \tag{6.8}$$

where $\mathbf{Z}' = [Z_1, Z_2, \ldots, Z_p]$,

$\mathbf{X}' = [X_1, X_2, \ldots, X_p]$,

$\boldsymbol{\mu}' = [\mu_1, \mu_2, \ldots, \mu_p]$ and

$$V^{1/2} = \begin{bmatrix} \sqrt{\sigma_{11}} & 0 & \cdots & 0 \\ 0 & \sqrt{\sigma_{22}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\sigma_{pp}} \end{bmatrix}$$

It follows that

$$E(\mathbf{Z}) = 0 \tag{6.9}$$

and

$$Cov(\mathbf{Z}) = (V^{1/2})^{-1} \Sigma (V^{1/2})^{-1} = \rho\ . \tag{6.10}$$

The principal components of $\mathbf{Z}$ are obtained from the eigenvectors of the correlation matrix ρ of $\mathbf{X}$. It is important to note that the eigenvalue-eigenvector value pairs $(\lambda_i, \mathbf{e_i})$ from ρ are generally not the same as the ones derived from Σ.

Result 6.3 (Johnson et al. (2002)) is essential in the **correlation matrix method** of principal component analysis.

Result 6.3

The i'th principal component of the standardized variables $\mathbf{Z}' = [Z_1, Z_2, \ldots, Z_p]$ with $Cov(\mathbf{Z}) = \rho$ is given by

$$Y_i = \mathbf{e_i}'\mathbf{Z} = \mathbf{e_i}'(V^{1/2})^{-1}(\mathbf{X} - \boldsymbol{\mu}) \qquad \text{for} \qquad i = 1,2,\ldots,p\ . \tag{6.11}$$

The eigenvalue-eigenvector pairs for ρ is denoted by

$$(\lambda_1, \mathbf{e_1}), (\lambda_2, \mathbf{e_2}), \ldots, (\lambda_p, \mathbf{e_p})$$

with

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p.$$

It also follows that:

$$\begin{aligned} &Total\,Variance \\ &\quad= \textstyle\sum_{i=1}^{p} Var(Z_i) \\ &\quad= \textstyle\sum_{i=1}^{p} Var(Y_i) \\ &\quad= p \end{aligned} \tag{6.12}$$

It follows that the proportion of the total population variance that is explained by the *k'*th principal component of $\mathbf{Z}$ is equal to:

$$\frac{\lambda_k}{p} \tag{6.13}$$

where the λ_k is the *k*th-largest eigenvalue of ρ.

The application of principal component analysis in the SAS window environment is discussed in Section 6.4.3.

6.4.2 The theoretical discussion of covariance matrices

Consider a random vector $\mathbf{X}' = [X_1, X_2, \ldots, X_p]$ of size $p \times 1$. The $p \times p$ population variance-covariance (or just covariance) matrix Σ is symmetric and has the following form:

$$\Sigma = Cov(X) = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{bmatrix}$$

where

σ_{ii} denotes the variance of X_i and

σ_{ij} denotes the covariance between X_i and X_j.

The sample variance-covariance (or just covariance) matrix S is used to estimate Σ . Suppose n observations of each the p variables $X_1, X_2, \ldots, X_p$ are made. Let

$$\bar{\mathbf{x}}' = \left[\bar{x}_{1,} \bar{x}_2, \ldots, \bar{x}_p\right]$$

denote the vector of sample averages constructed from the n observations.

The sample variance-covariance matrix S has the following form:

$$S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{n}\sum_{j=1}^{n}(x_{j1}-\bar{x}_1)^2 & \frac{1}{n}\sum_{j=1}^{n}(x_{j2}-\bar{x}_2)(x_{j1}-\bar{x}_1) & \cdots & \frac{1}{n}\sum_{j=1}^{n}(x_{jp}-\bar{x}_p)(x_{j1}-\bar{x}_1) \\ \frac{1}{n}\sum_{j=1}^{n}(x_{j1}-\bar{x}_1)(x_{j2}-\bar{x}_2) & \frac{1}{n}\sum_{j=1}^{n}(x_{j2}-\bar{x}_2)^2 & \cdots & \frac{1}{n}\sum_{j=1}^{n}(x_{jp}-\bar{x}_p)(x_{j2}-\bar{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n}\sum_{j=1}^{n}(x_{j1}-\bar{x}_1)(x_{jp}-\bar{x}_p) & \frac{1}{n}\sum_{j=1}^{n}(x_{j2}-\bar{x}_2)(x_{jp}-\bar{x}_p) & \cdots & \frac{1}{n}\sum_{j=1}^{n}(x_{jp}-\bar{x}_p)^2 \end{bmatrix}$$

Consider a SAS data set containing historical data values of certain market variables. Each column is a variable and each row (observation) contains a record of one data value for each variable. The sample variance-covariance matrix of the p variables is calculated by the SAS procedure ***Proc Corr***. The covariance matrix is written to a new SAS data set. The calculation and use of the sample variance-covariance matrix is discussed in the next section with reference to the case study.

6.4.3 Case study

The **modification** and **combination** of SAS data sets in the case study context are discussed in this section. The execution of **principal components analysis** and the estimation of a sample of a **variance-covariance matrix** are also discussed.

Program Code 6.18 is used to ensure that only observations with valid data values for the variable *Date* in *Market_History* SAS data set are used in further processing. *Market_History* is grouped in the *Riskdata* library.

Program Code 6.18: Modification of *Market_History*

```
Data RiskData.Market_History;
Set RiskData.Market_History;
Where Date is not missing;
Run;
```

The SAS procedure ***Proc Princomp*** is used to perform principal component analysis in the SAS window environment. The SAS data set, named *Yieldcurve_data* is specified in the *Data* option. The data set consists of historical data of the zero rates for different maturities. A statistical model may be fitted on each of the zero rates (see Chapter 8). The models are used to predict a future value for each risk factor variable. This forms the predicted future yield curve. These predicted values are perturbed by a series of simulated values in Monte Carlo simulation (see Chapter 10) that are used in the estimation of Value at Risk. The problem is that the movements in the zero rates for different maturities tend to be **highly correlated**. Thus, the negative and positive perturbations that are used in the simulation process may lead to **inaccurate results**. This is discussed in more detail in Section 10.2.7. The use of principal components analysis leads to a better solution. The first few principal components that explain a large proportion of the total variance are used in the modelling of a future yield curve. This is discussed in more detail in Chapter 8. Only the calculation of the principal components is discussed in this section.

The default method of *Proc Princomp* is the correlation matrix method. Principal components are calculated from the eigenvectors of the correlation matrix. The SAS data set specified in the *Out* option will contain the original data set (as specified in the *Data* option), as well as, the principal components that are obtained from the analysis. The SAS data set that is specified in the *Outstat* option will contain the means, standard deviations, number of observations, correlations, eigenvectors and eigenvalues from the analysis.

Program Code 6.19 is used to perform principal component analysis on the data set *Yieldcurve_data.*

Program Code 6.19: Principal Component Analysis

```
Proc Princomp Data=Riskdata.Yieldcurve_data noprint
Out = Work.PC_Scores
Outstat=Work.PC_Statistics;
Run;
```

A **scree plot** is a useful visual aid to determine an appropriate number of principal components. The eigenvalues are ordered from largest to smallest. A scree plot is a plot of the magnitude of the eigenvalue (y-axis) versus its number (x-axis). In order to determine the appropriate number of components, the user looks for an elbow or a bend in the scree plot. The scree plot of the eigenvalues of the correlation matrix, calculated in Program Code 6.19 is illustrated in Figure 6.7.

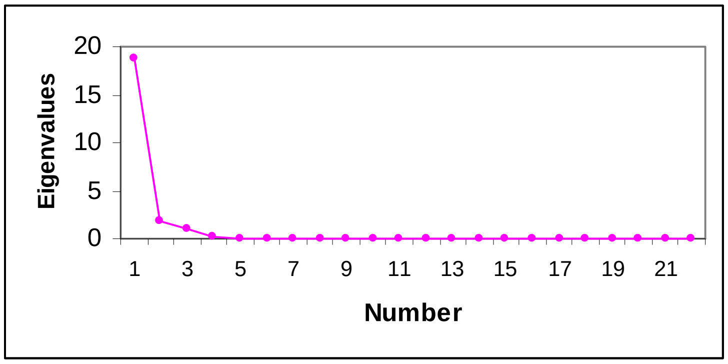


Figure 6.7: The scree plot of the eigenvalues

It follows from Figure 6.7 that the bend is after the second principal component. The historical data that is used in the analysis is of a relatively short length, namely only 127 observations. It may be that the third principal component may have a larger influence on the variance if the observations were more. The decision is thus made to use the first three principal components in subsequent analyses. These components explain 98.616% of the total population variance.

Program Code 6.20 is used to create new and update existing SAS data sets to contain the results from the execution of principal component analysis.

Program Code 6.20: The creation and modification of SAS data sets

```
/*The Date variable from Market_History and the first three principal
component variables from PC_Scores are used to create a new SAS data set
named Prindata in the Riskdata library.*/
Data Riskdata.Prindata (keep = Date Prin1 Prin2 Prin3);
Merge Riskdata.Market_History Work.PC_Scores;
Run;
/*The three principal component variables are added to the
Market_History data set, by using the Merge statement in the data
step.*/
Data Riskdata.Market_History;
Merge Riskdata.Market_History Riskdata.Prindata;
Run;
/*The Current_Market SAS data set are created from the last observation
of the Market_History SAS data set. The observation contains the market
information of the valuation date. */
Data RiskData.Current_Market;
     Set RiskData.Market_History;
     If _N_ = 127 then Output;
Run;
```

Program Code 6.21 continues …

```
/*The eigenvectors of the first three principal components are stored in
the Eigenvectors SAS data set in the Riskdata library.*/
Data Riskdata.Eigenvectors(Drop =_Type_  );
Set PC_Statistics;
Where _Name_ in ("Prin1" "Prin2" "Prin3");
Run;
/*The mean and standard deviation of each variable in Yieldcurve_data
are stored in the Mean_Std SAS data set in the Riskdata library. The
information is obtained from the PC_Statistics data set.*/
Data Riskdata.Mean_Std (Drop=_Type_ _Name_);
Set Work.PC_statistics;
Where _Type_ in ("MEAN" "STD");
Run;
```

The SAS procedure named *Proc Corr* is used in Program Code 6.21 to calculate a sample variance-covariance matrix from historical market data values. The variance-covariance matrix is stored in a SAS data set with name *Market_Covar* in the *Riskdata* library.

Program Code 6.21: The creation of a covariance matrix in a SAS data set

```
/*The SAS data set Market_History in the Work library is created from
the SAS data set with the same name in the Riskdata library.  The
variable with name Date is dropped.*/
Data Work.Market_History(Drop = Date);
Set Riskdata.Market_History;
Run;
/*Proc Corr is used to create a sample variance-covariance matrix.  The
matrix is stored with other observations in the SAS data set that is
specified in the Outp option.  The matrix is calculated from the SAS
data set that is specified in the Data option. The Cov option in the
Proc statement is necessary in the creation of the variance-covariance
matrix.*/
Proc Corr Cov noprint
Data= Work.Market_History
Outp= Work.Market_Cov;
Run;
/*The variance-covariance matrix part of the SAS data set Market_Cov is
written to the new SAS data set Market_Covar in the Riskdata library.*/
Data RiskData.Market_Covar;
Set Market_Cov;
Where _Type_ = "COV";
Run;
```

The resulting SAS data sets are used in further analyses in Risk Dimensions.

6.5 Data-driven registration

6.5.1 General

The Risk Dimensions variables that are registered in a risk environment are divided into two groups in Section 5.1. The first group are the variables that are contained in SAS data sets like *Tradebook, Bondbook, Swapbook, Market_History* and *Yieldcurve_data*. The second group are the variables that are not contained in any SAS data sets. **Data-driven registration** is an alternative method to register the first group of variables. It is a method that registers Risk Dimensions variables from the variable or column names in SAS data sets. This method works very well for case studies where the number of variables in the SAS data sets is large. The variables in the second group that are necessary in the risk environment are still registered by *Proc Risk* as described in Section 5.2.

The variables that are necessary in the *Casestudy_Env* risk environment is listed in Table 5.1. Each variable has a value of either *1* or *2* for the *Group* column. The following two methods of variable registration are available:

1. The registration of the variables of both groups by using *Proc Risk* (Chapter 5).

2. The registration of the variables in group 1 by data-driven registration (this section) and the registration of the variables in group 2 by using *Proc Risk* (Chapter 5).

The data-driven registration process consists of two parts. In the first part, the SAS procedure, *Proc Contents* and the descriptor portion of the SAS data set are used to create a new SAS data set. This new SAS data set is called a **variable definition data set** and contains the variable names and attributes of the original

SAS data set. Various variable definition data sets are created from the first variable definition data set and contain specific information about each attribute of each variable. In the second part of the process the information contained in the variable definition data sets is **read** into SAS Risk Dimensions and the variables are registered appropriately in the risk environment.

6.5.2 The creation of variable definition data sets

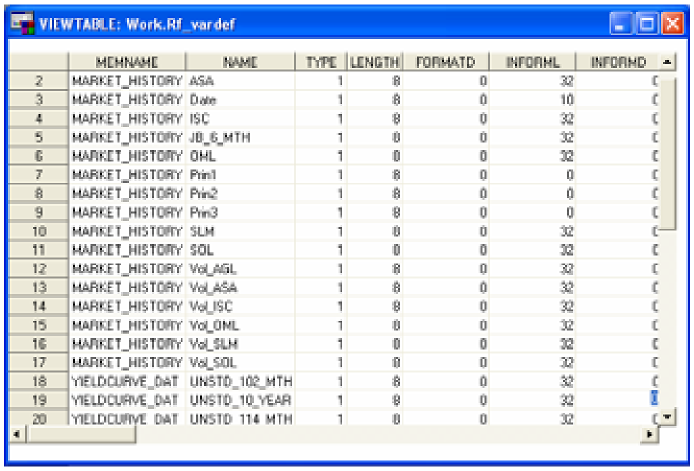
The first step in data-driven registration, is to create a SAS data set called a **variable definition data set** that contains all the relevant names and attributes for the variables contained in the original SAS data set. If the case study is considered, a variable definition data set is created for each of the following SAS data sets, *Tradebook, Bondbook, Swapbook, Market_History* and *Yieldcurve_data*.

Program Code 6.22 is used to create a variable definition data set for the risk factor variables. The SAS procedure *Proc Contents* is used to create the variable definition data sets. The *Market_History* and *Yieldcurve_data* data sets that are stored in the *Riskdata* library are specified in the *Data* option of the separate procedures. The variable definition data sets are specified in the *Out* option. The data set is *Rf_Vardef* and is grouped in the *Work* library.

Program Code 6.22: The creation of the variable definition data set *Rf_Vardef*

```
/*Proc Contents is used to create the variable definition data sets as
specified in the Out option. The name of the original SAS data set is
specified in the Data option. */
Proc Contents Data = RiskData.Market_History
              Out  = Work.Rf_vardef1 ;
Run;
Proc Contents Data = Riskdata.Yieldcurve_data
              Out  = Work.Rf_vardef2;
Run;
/*Concatenation is used to combine the two variable definition data sets
created by Proc Contents into a single data set*/
Data Work.Rf_vardef;
Set Work.Rf_vardef1 Work.Rf_vardef2;
Run;
```

Each variable of the SAS data sets *Market_History* and *Yieldcurve_data*, together with its attributes is recorded as a row in the SAS data set *Rf_Vardef*. The most important attributes are *name, type* and *length*. The data portion of the *Rf_Vardef* SAS data set is viewed in Figure 6.8.


VIEWTABLE: Work.Rf_vardef

	MEMNAME	NAME	TYPE	LENGTH	FORMATD	INFORML	INFORMD
2	MARKET_HISTORY	ASA	1	8	0	32	C
3	MARKET_HISTORY	Date	1	8	0	10	C
4	MARKET_HISTORY	ISC	1	8	0	32	C
5	MARKET_HISTORY	JB_6_MTH	1	8	0	32	C
6	MARKET_HISTORY	OML	1	8	0	32	C
7	MARKET_HISTORY	Prin1	1	8	0	0	C
8	MARKET_HISTORY	Prin2	1	8	0	0	C
9	MARKET_HISTORY	Prin3	1	8	0	0	C
10	MARKET_HISTORY	SLM	1	8	0	32	C
11	MARKET_HISTORY	SOL	1	8	0	32	C
12	MARKET_HISTORY	Vol_AGL	1	8	0	32	C
13	MARKET_HISTORY	Vol_ASA	1	8	0	32	C
14	MARKET_HISTORY	Vol_ISC	1	8	0	32	C
15	MARKET_HISTORY	Vol_OML	1	8	0	32	C
16	MARKET_HISTORY	Vol_SLM	1	8	0	32	C
17	MARKET_HISTORY	Vol_SOL	1	8	0	32	C
18	YIELDCURVE_DAT	UNSTD_102_MTH	1	8	0	32	C
19	YIELDCURVE_DAT	UNSTD_10_YEAR	1	8	0	32	C
20	YIELDCURVE_DAT	UNSTD_114_MTH	1	8	0	32	C

Figure 6.8: The *Rf_Vardef* SAS data set

The third row of this data set, containing the variable attributes of *Date* is not necessary for further analysis. Program Code 6.23 is used to modify the variable definition data set with name *Rf_Vardef* so that the third row is excluded.

Program Code 6.23: The modification of the *Rf_Vardef* SAS data set

```
Data Work.Rf_vardef;
Set Work.Rf_vardef;
Where Name not in ("Date");
Run;
```

Program Code 6.24 to Program Code 6.26 is used to convert the attribute information in the variable definition data set *Rf_Vardef* into more specific variable definition data sets that are stored in the permanent library, ***Local***. Program Code 6.24 is used to create a variable definition set that contains information about the risk factor variables that need to be registered in the risk environment.

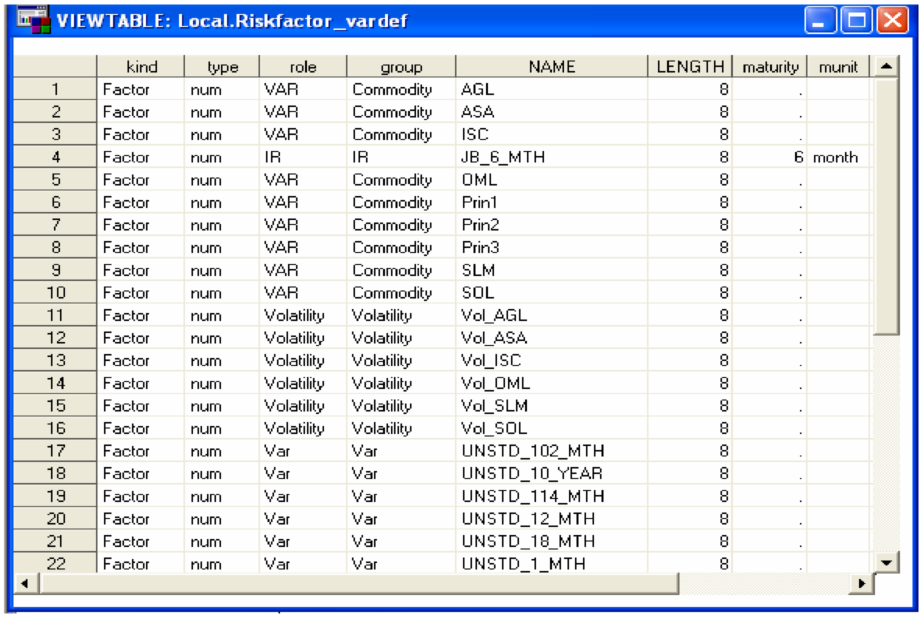
Program Code 6.24: The creation of a variable definition set for risk factor variables

```
/*A variable definition data set, Riskfactor_vardef is created in the
SAS library Local.  The keep option specifies the variables that are
kept in this data set.*/
Data Local.Riskfactor_vardef (Keep = Name Kind Type Length Munit Mlevel
                              Role Maturity Currency Group Refname1
                              Label);
/*The following statement specifies the maximum length of the data
values of the new variables*/
Length Kind Type Role Group Mlevel $ 15;
/*The Set statement specifies that the input SAS data set Rf_vardef is
grouped in the Work library.  The Rename option specifies that the
variable, named Type in Rf_Vardef is renamed to Sastype in
Riskfactor_vardef.*/
Set Rf_Vardef(Rename = (Type = Sastype));
/*A new variable named Kind is created. Each data value in this variable
is the character string Factor*/
Kind = "Factor";
/*Conditional logic is used to assign the data type attribute of each
variable*/
If Sastype = 1 then type = "num";
Else if Sastype = 2 then type = "char";
/*The names of the variables are used with conditional logic to assign
attributes to certain variables, for example JB_6_MTH */
If Scan(name,1,"_") = "JB" then do;
Role = "IR";
Maturity = Input(scan(name,2,"_"),8.);
Munit = "month";
Currency = "ZAR";
Group = "IR";
Label = "The JIBAR    Month Yield";
Substr(label,11,2)=maturity;
End;
Else if upcase(scan(name,1,"_")) = "VOL" then do;
Role = "Volatility";
Group = "Volatility";
Refname1 = Scan(name,2,"_");
Label= "The volatility of";
Substr(label,20,4) = Scan(name,2,"_");
End;
Else if Upcase(scan(name,1,"_")) = "UNSTD" then do;
Role = "Var";
Group ="Var";
Label = "         Unstd Zero Rate";
Substr(label,1,4) = Scan(name,2,"_");
Substr(label,4,5) = Scan(name,3,"_");
End;
```

Program Code 6.24 continues …

```
Else do;
/*Not one of the above*/
Role = "VAR";
Group = "Commodity";
Mlevel = "Interval";
Label = "The generic risk factor ";
Substr(label,25,4) = Scan(name,1,"_");
End;
Run;
```

The resulting variable definition data set by name *Riskfactor_vardef* is viewed in Figure 6.9.



	kind	type	role	group	NAME	LENGTH	maturity	munit
1	Factor	num	VAR	Commodity	AGL	8	.	
2	Factor	num	VAR	Commodity	ASA	8	.	
3	Factor	num	VAR	Commodity	ISC	8	.	
4	Factor	num	IR	IR	JB_6_MTH	8	6	month
5	Factor	num	VAR	Commodity	OML	8	.	
6	Factor	num	VAR	Commodity	Prin1	8	.	
7	Factor	num	VAR	Commodity	Prin2	8	.	
8	Factor	num	VAR	Commodity	Prin3	8	.	
9	Factor	num	VAR	Commodity	SLM	8	.	
10	Factor	num	VAR	Commodity	SOL	8	.	
11	Factor	num	Volatility	Volatility	Vol_AGL	8	.	
12	Factor	num	Volatility	Volatility	Vol_ASA	8	.	
13	Factor	num	Volatility	Volatility	Vol_ISC	8	.	
14	Factor	num	Volatility	Volatility	Vol_OML	8	.	
15	Factor	num	Volatility	Volatility	Vol_SLM	8	.	
16	Factor	num	Volatility	Volatility	Vol_SOL	8	.	
17	Factor	num	Var	Var	UNSTD_102_MTH	8	.	
18	Factor	num	Var	Var	UNSTD_10_YEAR	8	.	
19	Factor	num	Var	Var	UNSTD_114_MTH	8	.	
20	Factor	num	Var	Var	UNSTD_12_MTH	8	.	
21	Factor	num	Var	Var	UNSTD_18_MTH	8	.	
22	Factor	num	Var	Var	UNSTD_1_MTH	8	.	

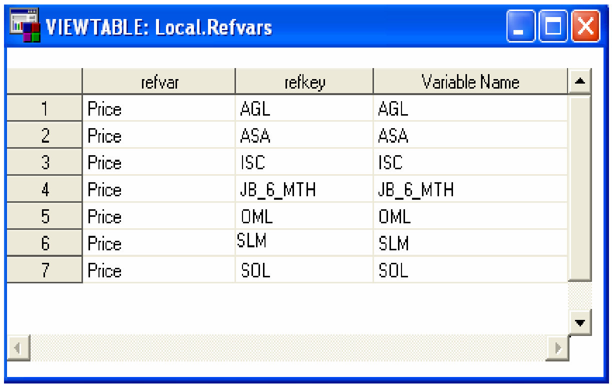
Figure 6.9: The *Riskfactor_vardef* SAS data set

Program Code 6.25 is used to create a variable definition data set that contains information about the risk factor variable attribute *Refmap* for certain variables.

Program Code 6.25: The creation of a variable definition set for reference variables

```
/*A variable definition data set, Refvars is created in the SAS library
Local.  The Keep option specifies the variables that are kept in this
data set.*/
Data Local.Refvars (Keep = Name Refvar Refkey);
/*The following statement specifies the maximum length of the data
values of the new variables*/
Length Refvar $32
       Refkey $20;
/*The Set statement specifies that the input SAS data set is Rf_vardef
that is stored in the Work library.*/
Set Rf_vardef;
/*The Where statement specifies which rows are included in the new SAS
data set*/
Where Name in ("SOL" "AGL" "SLM" "ISC" "OML" "ASA" "JB_6_MTH");
Refvar = "Price";
Refkey = Name;
Run;
```

The resulting variable definition data set with name *Refvars* is viewed in Figure 6.10.



	refvar	refkey	Variable Name
1	Price	AGL	AGL
2	Price	ASA	ASA
3	Price	ISC	ISC
4	Price	JB_6_MTH	JB_6_MTH
5	Price	OML	OML
6	Price	SLM	SLM
7	Price	SOL	SOL

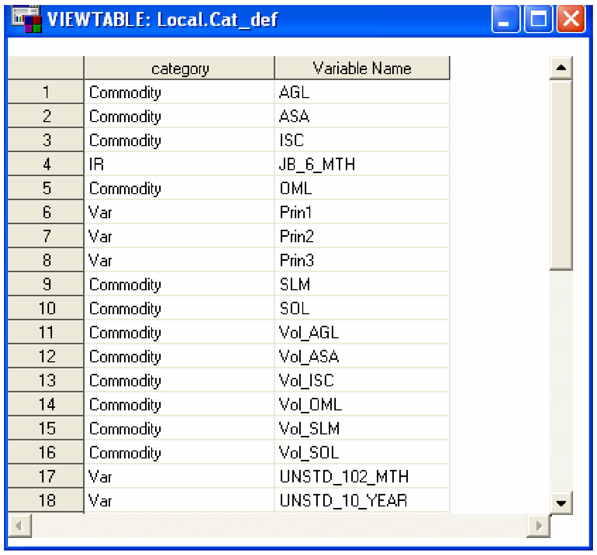
Figure 6.10: The *Refvars* SAS data set

A variable definition data set that contains information about the *Category* attribute for each risk factor variable, is created. This attribute is used in the calculation of marginal and conditional Value at Risk. Program Code 6.26 creates the variable definition data set *Cat_Def* in the library *Local*:

Program Code 6.26: The creation of a variable definition set for risk factor attribute with name *categories*

```
/*A variable definition data set, Cat_def is created in the SAS library
Local.  The keep option specifies the variables that are kept in this
data set.*/
Data Local.Cat_def (Keep = Name Category);
/*The following statement specifies the maximum length of the data
values of the new variables*/
Length Category $32;
Set Rf_vardef;
If Upcase(Scan(name,1,"_")) = "JB" then Category = "IR";
Else if Upcase(Scan(name,1,"_")) = "UNSTD" then Category = "Var";
Else if Upcase(Substr(name,1,4)) = "PRIN" then Category = "Var";
Else if Upcase(Scan(name,1,"_")) = "VOL" then Category = "Volatility";
Else do;
Category = "Commodity";
End;
Run;
```

The structure of the SAS data set *Cat_Def* is illustrated in Figure 6.11.



	category	Variable Name
1	Commodity	AGL
2	Commodity	ASA
3	Commodity	ISC
4	IR	JB_6_MTH
5	Commodity	OML
6	Var	Prin1
7	Var	Prin2
8	Var	Prin3
9	Commodity	SLM
10	Commodity	SOL
11	Commodity	Vol_AGL
12	Commodity	Vol_ASA
13	Commodity	Vol_ISC
14	Commodity	Vol_OML
15	Commodity	Vol_SLM
16	Commodity	Vol_SOL
17	Var	UNSTD_102_MTH
18	Var	UNSTD_10_YEAR

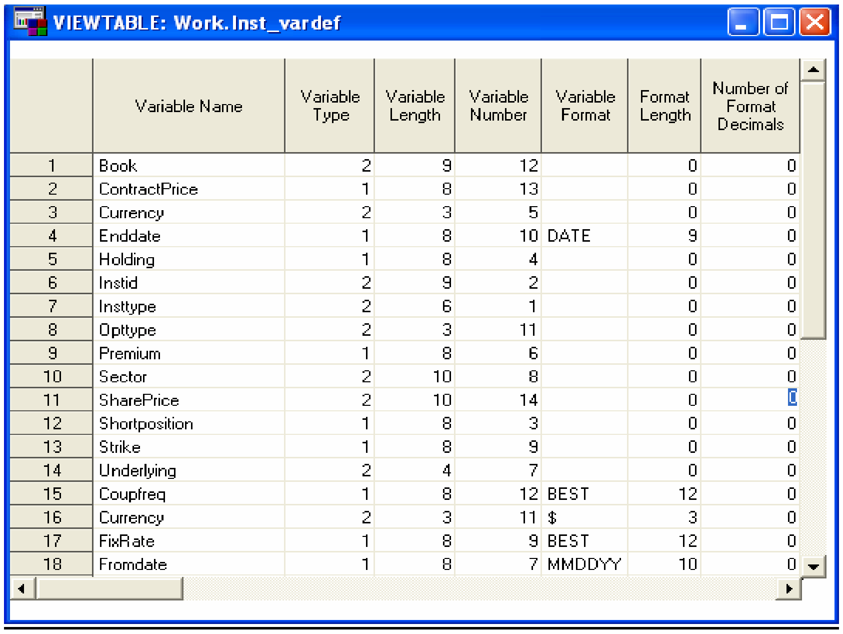
Figure 6.11: The *Catdef* SAS data set

Program Code 6.27 is used to create a variable definition data set *Inst_vardef* that contains information about the instrument variables that need to be registered in the risk environment. The SAS procedure *Proc Contents* is used to create variable definition data sets for the SAS data sets *Tradebook, Swapbook* and *Bondbook* respectively. The *Set* statement in the Data step is used to combine these data sets to create *Inst_vardef.*

Program Code 6.27: The creation of variable definition data sets for the instruments variables

```
/*Proc Contents is used to create the variable definition data sets as
specified in the Out option. The name of the original SAS data set is
specified in the Data option. */
Proc Contents Data=RiskData.TradeBook Out = Work.Inst_vardef1 ;
Run;
Proc Contents Data=RiskData.Swapbook Out = Work.Inst_vardef2 ;
Run;
Proc Contents Data=RiskData.Bondbook Out = Work.Inst_vardef3 ;
Run;
/*The Data step is used to modify the Inst_Vardef2 and Inst_Vardef3 SAS
data sets*/
Data Work.Inst_vardef2;
Set Work.Inst_vardef2;
Where Upcase(name) not in ("BOOK" "PREMIUM" "MATURITYDATE");
Run;
Data Work.Inst_vardef3;
Set Work.Inst_vardef3;
Where Upcase(name) not in ("BOOK" "COUPFREQ" "NOTIONAL" "PREMIUM");
Run;
/*The three variable definition data sets are combined into one variable
definition data set in the following data step*/
Data Work.Inst_vardef;
Set Work.Inst_vardef1 Work.Inst_vardef2 Work.Inst_vardef3;
Run;
```

Each variable of the SAS data sets *Tradebook*, *Bondbook* and *Swapbook*, together with, its attributes, is recorded as a row in the SAS data set *Inst_Vardef*. The most important attributes are *name, type* and *length*. Figure 6.12 is used to view the data portion of the *Inst_Vardef* data set.



VIEWTABLE: Work.Inst_vardef

	Variable Name	Variable Type	Variable Length	Variable Number	Variable Format	Format Length	Number of Format Decimals
1	Book	2	9	12		0	0
2	ContractPrice	1	8	13		0	0
3	Currency	2	3	5		0	0
4	Enddate	1	8	10	DATE	9	0
5	Holding	1	8	4		0	0
6	Instid	2	9	2		0	0
7	Insttype	2	6	1		0	0
8	Opttype	2	3	11		0	0
9	Premium	1	8	6		0	0
10	Sector	2	10	8		0	0
11	SharePrice	2	10	14		0	0
12	Shortposition	1	8	3		0	0
13	Strike	1	8	9		0	0
14	Underlying	2	4	7		0	0
15	Coupfreq	1	8	12	BEST	12	0
16	Currency	2	3	11	$	3	0
17	FixRate	1	8	9	BEST	12	0
18	Fromdate	1	8	7	MMDDYY	10	0

Figure 6.12: The *Work.Inst_vardef* SAS data set

Program Code 6.28 is used to convert the attribute information in the variable definition data set *Inst_Vardef* into a more specific variable definition data set, named *Instrument_Vardef* that is stored in the permanent library, *Local*.

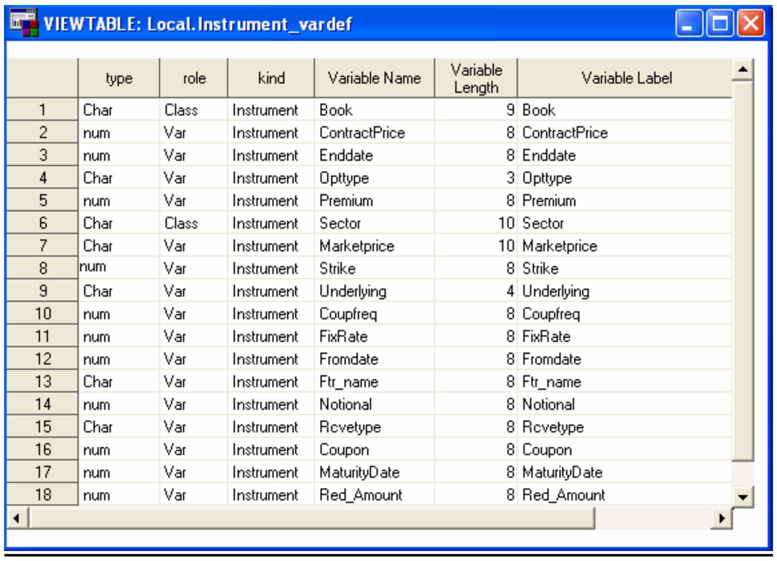
Program Code 6.28: The creation of a variable definition set for instrument variables

```
/*A variable definition data set, Instrument_vardef is created in the
SAS library Local.  The variables that are kept in this data set are
specified in the Keep option.*/
Data Local.Instrument_vardef(keep = Name Kind Type Length Role Label);
/*The following statement specifies the maximum length of the data
values of the new variables*/
Length Type $12 Role $18 Kind $15;
/*The Set statement specifies that the input SAS data set is Inst_vardef
that is grouped in the Work library.  The Rename option specifies that
the variable Type in the Inst_vardef is renamed to Sastype in
Instrument_vardef.*/
Set Inst_vardef(Rename = (Type = Sastype));
/*A new variable with name Kind is created. Each data value of this
variable is the character string Instrument*/
Kind = "Instrument";
/*Conditional logic is used to assign the data type attribute of each
variable*/
If Sastype = 1 then Type = "num";
Else if Sastype = 2 then Type = "char";
```

Program Code 6.28 continues …

```
/*The  data  value  of  Shareprice  for  the  name  variable  is  changed  to
MarketPrice*/
If Upcase(Name) = "SHAREPRICE" then Name = "Marketprice";
/*A classification role is assigned to Sector and Book*/
If Name in ("Sector" "Book") then Role = "Class";
Else do;
Role = "Var";
End;
Label = name;
/*The Where statement specifies which rows are included in the new SAS
data set*/
Where Upcase(Name) not in ("INSTTYPE" "CURRENCY" "HOLDING"
                        "MATURITYDATE"  "SHORTPOSITION" "INSTID");
Run;
```

The data portion of the *Inst_vardef* SAS data set is viewed in Figure 6.14.



VIEWTABLE: Local.Instrument_vardef

	type	role	kind	Variable Name	Variable Length	Variable Label
1	Char	Class	Instrument	Book	9	Book
2	num	Var	Instrument	ContractPrice	8	ContractPrice
3	num	Var	Instrument	Enddate	8	Enddate
4	Char	Var	Instrument	Opttype	3	Opttype
5	num	Var	Instrument	Premium	8	Premium
6	Char	Class	Instrument	Sector	10	Sector
7	Char	Var	Instrument	Marketprice	10	Marketprice
8	num	Var	Instrument	Strike	8	Strike
9	Char	Var	Instrument	Underlying	4	Underlying
10	num	Var	Instrument	Coupfreq	8	Coupfreq
11	num	Var	Instrument	FixRate	8	FixRate
12	num	Var	Instrument	Fromdate	8	Fromdate
13	Char	Var	Instrument	Ftr_name	8	Ftr_name
14	num	Var	Instrument	Notional	8	Notional
15	Char	Var	Instrument	Rcvetype	8	Rcvetype
16	num	Var	Instrument	Coupon	8	Coupon
17	num	Var	Instrument	MaturityDate	8	MaturityDate
18	num	Var	Instrument	Red_Amount	8	Red_Amount

Figure 6.14: The *Local.Inst_vardef* SAS data set

6.5.3 Registering variables from variable definition data sets

A special set of program statements are used to read the variable information from the variable definition data sets and to register these variables in the specified risk environment.

The program statements are:

Readvars

It registers instrument variables and risk factor variables.

Readcategories

It registers the risk factor variable attribute, named *Category*, that is used in marginal and conditional simulation.

Readrefs

It registers reference variables and reference keys. The reference variables and reference keys have to be specified in the risk factor variable attribute *Refmap*.

Program Code 6.29 uses the information contained in the variable definition data sets that were created in Program Code 6.22 to Program Code 6.28, to register the appropriate variables in the *Casestudy_Env* risk environment.

Program Code 6.29: The registration of variables in the risk environment

```
Proc Risk;
/*The Environment statement is used to create a new environment, named
Casestudy_Env is created in the folder C:\Risk_Warehouse\Env */
Environment new = "&RiskEnv";
/*The appropriate Read statement is linked to the appropriate variable
definition data set to register the correct variables*/
Readvars Data = Local.Riskfactor_vardef;
Readrefs Data = Local.Refvars;
Readcategories Data = Local.Cat_def;
Readvars Data = Local.Instrument_vardef;
/*The  Environment  statement  is  used  to  save  the  Casestudy_Env
environment*/
Environment save;
Run;
```

6.6 Summary

The first important topic that was discussed in this chapter is data preparation in the SAS window environment. The modification and combination of SAS data sets were used in the data preparation methods. The resulting SAS data sets are used in risk analyses. Other SAS concepts like the *Where* and *If* statements, SAS arrays and Do loops were also discussed. These concepts will also be used in subsequent chapters. A theoretical discussion of principal component analysis and variance-covariance matrices, respectively were also included. The calculation of principal components and a sample variance-covariance matrix for the case study was executed in the SAS environment. Lastly an alternative variable registration method, namely data-driven registration was discussed with reference to the case study.

7

METHOD PROGRAMS AND INSTRUMENT TYPES

7.1 Introduction

The registration of SAS Risk Dimensions variables in a risk environment was illustrated in Chapter 5. The Risk Dimensions structures that are created next are **method programs** and **instrument types**.

A **method program** is a block of program code that is used to directly access and change the values of the registered Risk Dimensions variables. The three different types of method programs that are discussed in detail are:

- Pricing methods,
- Instrument input methods and
- Risk factor transformation methods.

A **pricing method** is used to calculate the value of a financial instrument and assign the value to the system-defined variable _*Value*_. An **instrument input method** is used to calculate data values for instrument variables from the data values of other instrument variables. A **risk factor transformation method** is used to calculate data values for risk factor variables from the data values of other risk factor variables.

An **instrument type** plays the same role in Risk Dimensions as a financial security in the real world. Examples are instrument types with names such as *Equity, Future, Bond, Option* and *Int_Swap*. A **list** of **variables** that is necessary to group and value the instrument, as well as a **pricing method** is specified for each instrument type. An instrument input method may be used optionally.

Method programs make use of **SAS subroutines** and user-defined **SAS functions** to ensure that program code that is frequently used does not have to be re-written in each of the different method programs. SAS subroutines and user-defined SAS functions are callable blocks of program code and have input and output parameters.

The SAS procedure, named ***Proc Compile*** is used to create method programs, SAS subroutines and user-defined SAS functions and is discussed in Section 7.2. The ***Instrument*** statement in *Proc Risk* is used to create instrument types.

The **steps** that are necessary to create the above mentioned Risk Dimensions structures may be summarized as follows:

1. The necessary SAS **subroutines** and user-defined SAS **functions** are created with *Proc Compile* (see Section 7.3).

2. The necessary **method programs** are created with *Proc Compile*. The subroutines and functions of Step 1 are used within the method programs (see Section 7.4).

3. An **instrument type** is created for each type of financial instrument in the trade book or portfolio with the *Instrument* statement (see Section 7.5). A list of Risk Dimensions variables and a pricing method is required for each instrument type.

7.2 The SAS procedure *Proc Compile*

The SAS procedure by name of *Proc Compile* is used to create SAS subroutines, user-defined SAS functions and method programs that are used in Risk Dimensions. The general form of the *Proc Compile* procedure is illustrated in Program Code 7.1.

Program Code 7.1: The general form of the *Proc Compile* procedure

```
Proc Compile Env = "Environment-name"
             Outlib = "Catalog-name"
             Package = "Package-name";
Statements that create Subroutines, Functions or Method Programs;
Run;
```

The *Env* option is used to specify the name and physical location of the SAS catalog that contains the objects of the risk environment in which the user-defined functions, subroutines and method programs are created and used. The SAS catalog is stored as a SAS file on the hard drive. The *Outlib* option is used to specify the SAS catalog where the user-defined functions, subroutines and method programs are stored. The name and location of the SAS catalog that is specified in the *Env* option is usually also specified in the *Outlib* option. The user-defined functions, subroutines and method programs are thus stored in the SAS catalog that also contains the risk environment that they are used in. A package is defined as a collection of related subroutines or function. The *Package* option is used to specify a name for the package in which the compiled user-defined functions, method programs and subroutines are stored. The statements that are used to create the method programs, subroutines or functions are listed after the *Proc Compile* statement and before the *Run* statement.

Various other options may also be specified in the *Proc Compile* statement, but are not discussed in this document. The options are however useful in modifying the structure in which the functions are stored and grouped. The use of *Proc Compile* is illustrated in Sections 7.3 and 7.4.

7.3 Subroutines and Functions

SAS subroutines and user-defined **SAS functions** are used in method programs to ensure that program code that is frequently used does not have to be re-written in each of the different method programs.

A **SAS subroutine** is a block of program code that may have multiple input and output parameters. Subroutines are created in the *Proc Compile* procedure (see Section 7.2) and are called or executed in method programs. The ***Subroutine*** statement is used to declare a SAS subroutine. The *Endsub* statement ends the subroutine declaration. The *Group* option, groups similar subroutines together within the package, as specified in *Package* option in the *Proc Compile* statement. The *Outargs* statement specifies all the variables that are used as output parameters. The rest of the variables specified in the *Subroutine* statement are used as input parameters. A subroutine is executed in a method program by using the ***Call*** statement. The execution is illustrated in Section 7.4.

Consider the case study again. The yield curve, named *Zerocurve* consists of a series of risk-free interest rate values for corresponding time to maturity values. The yield curve is used in the calculation of an estimate of the risk-free rate of interest r for a given time to maturity t. The following linear interpolation function is used in the determination of r for a given t:

$$\begin{aligned} r &= i_1 && \text{for} \quad t \le m_1 \\ r &= i_j + (i_{j+1} - i_j) \times \frac{t - m_j}{m_{j+1} - m_j} && \text{for} \quad m_j \le t \le m_{j+1} \\ r &= i_n && \text{for} \quad t \ge m_n \end{aligned} \qquad (7.1)$$

where

i_j = the interest rate value of the j'th element of the yield curve,

m_j = the maturity value of the j'th element of the yield curve,

n = the number of elements in the yield curve.

A SAS subroutine with name *RF_Interp* is created and is used in the *Casestudy_Env* risk environment. The input parameter, named *Time* is defined in terms of a fraction of a year. The *spotval[*]* input parameter denotes a SAS array that consists of risk-free interest rates values. The input parameter *maturity[*]* contains the corresponding time to maturity of each of the risk-free interest rates. The variable *RF_Rate* is an output parameter and contains the value of the risk-free rate of interest that corresponds to the time to maturity of the variable *Time*. The variables that are used in a subroutine do not have to be defined in the risk environment. The creation of the *RF_Interp* subroutine that uses (7.1) is illustrated in Program Code 7.2.

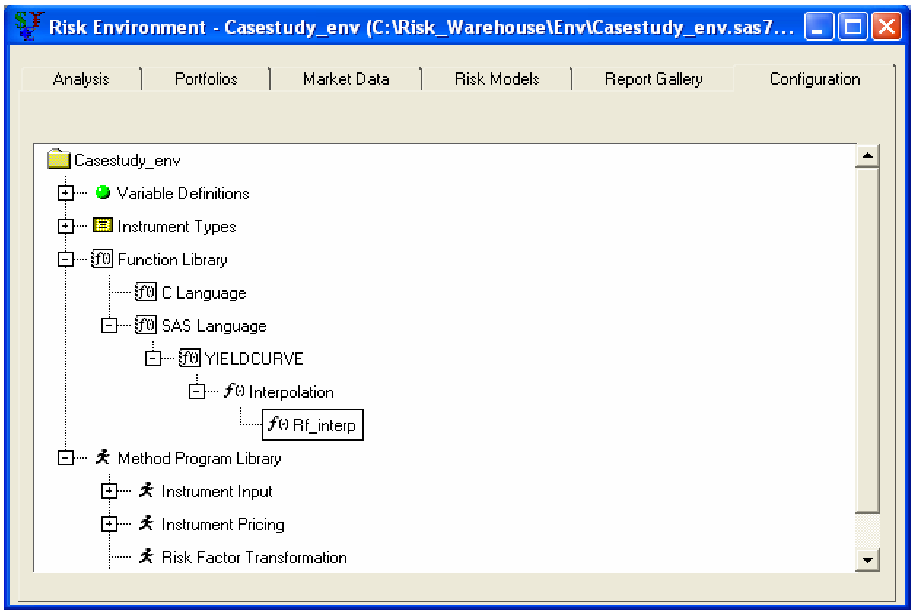
Program Code 7.2: The *RF_Interp* subroutine

```
/*The SAS procedure, Proc Compile is used to create a subroutine that is
defined in the Casestudy_Env risk environment. The subroutine is also
saved in the SAS catalog that contains the Casestudy_Env environment and
is grouped in the package named Yieldcurve*/
Proc Compile Env = "&RiskEnv" Outlib = "&RiskEnv" Package = Yieldcurve;
/*The name of the subroutine is RF_Interp, a label is specified and the
subroutine is grouped under the Interpolation kind of subroutines and
functions in the Yieldcurve package*/
Subroutine RF_Interp(Time,Spotval[*],Maturity[*],RF_Rate)
      Label = "Risk-free rate of interest" Group = "Interpolation";
/*The variable RF_Rate is defined as an output parameter*/
Outargs RF_Rate;
/*The Dim function assigns the value of the number of elements in the
SAS array to the variable named Npoint. */
Npoint = Dim(spotval);
/*The following If and Do statements are used to derive a value for the
risk-free rate of interest for a specified time to maturity, using
linear interpolation as illustrated in (7.1)*/
If (time <= maturity{1}) then spotrate = spotval{1};
If (time >= maturity{npoint}) then spotrate = spotval{npoint};
```

Program Code 7.2 continues…

```
If (time > maturity{1}) and (time < maturity{Npoint}) then
  Do;
        Do j = 1 to npoint-1;
        If(time > maturity{j}) and (time <= maturity{j+1}) then
            do;
            spotrate= spotval{j} + (( spotval{j+1} - spotval{j})
                      * (time - maturity{j})
                      / (maturity{j+1} - maturity{j}));
            End; /*The third do*/
        End;  /*The second do*/
  End;  /*The first do*/
/*The output parameter RF_Rate is assigned to contain the value of the
variable Spotrate that contains the calculated risk-free rate of
interest*/
RF_Rate = spotrate;
Endsub;
Run;
```

The SAS subroutine named *RF_Interp* is viewed under the *Function library* option in the *Configuration* tree in the GUI. This subroutine is created in a SAS program and is thus, listed under the *SAS language* heading. The package is specified as *Yieldcurve*. The subroutine is grouped in the package under the *Interpolation* group. Figure 7.1 illustrates the grouping of the subroutine.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy_env.sas7...

Analysis | Portfolios | Market Data | Risk Models | Report Gallery | Configuration

```
Casestudy_env
  + Variable Definitions
  + Instrument Types
  - Function Library
      C Language
    - SAS Language
      - YIELDCURVE
        - Interpolation
            Rf_interp
  - Method Program Library
    + Instrument Input
    + Instrument Pricing
      Risk Factor Transformation
```

Figure 7.1: The grouping of the SAS subroutine *RF_Interp*

A **user-defined SAS function** is a special case of a SAS subroutine. It has multiple input parameters, but only one output parameter that is returned. The ***Function*** statement in the procedure *Proc Compile* is used to declare a user-defined SAS function. The *Endsub* statement ends the function declaration. The *Kind* option specifies the kind of function that is being declared and it is usually one of *Price, Input, Trans* or *Kind*. Other user-defined kinds may also be specified.

Program Code 7.2 may be modified to change the subroutine *RF_Interp* into a user-defined SAS function with the same name. The *Subroutine* statement is replaced with the *Function* statement and the *Group* option with the *Kind* option. A *Return* statement that returns the value of the output parameter *Rf_Rate* is added before the *Endsub* statement. The user-defined function *RF_Interp* will have the same use as the subroutine with the same name.

The subroutine and user-defined functions that were discussed above are both created in SAS programs. SAS Risk Dimensions offers, in addition the facility to import functions that are created in the programming languages C or C^{++}. This enables the creation and use of more complicated subroutines and pricing functions that may be very useful in practical applications. The importation of C functions is an advanced topic and is not covered in this document.

7.4 Method programs

7.4.1 General

A **method program** is a block of program code that is used to directly access and change the values of the registered Risk Dimensions variables. This program code is used to calculate data values for registered variables that are not observed in the SAS data sets that contain market and position information. The data values of the registered variables that are observed in these data sets are, however, used in the calculations.

The six different **kinds** of method programs that are available are:

- Instrument input methods
- Pricing methods,
- Risk factor transformation methods,
- Postpricing methods,
- Postvar methods and
- Project methods.

A **pricing method** is assigned to each instrument type. It is used to calculate the value of each open position taken in the instrument type. The calculated value is assigned to the system-defined variable, named *_Value_*. An **instrument input method** is used to calculate data values for instrument variables, based on the data values of other instrument variables. A **risk factor transformation method** uses the data values of some risk factor variables to calculate data values for other risk factor variables. These three kinds of method programs are discussed in detail in Sections 7.4.2 to 7.4.4. The other three kinds of method programs are mentioned briefly in Section 7.4.5.

The creation of method programs and the use of subroutines, functions and variable suffixes in it, are discussed in the rest of this section.

The ***Method*** statement in *Proc Compile* starts a method block and the *Endmethod* statement ends the block. The *Kind* option in the *Proc* statement specifies the kind of method that is being declared. The available options in *Kind* are: *Input*, *Price*, *Trans*, *Postprice*, *Postvar* and *Project*.

Method programs use subroutines or user-defined functions to ensure that frequently used program code is not repeated. A **subroutine** is **executed** in a method program by the ***Call*** statement.

The *Call* statement has the following general form:

```
Call Subroutine-name(variable1,variable2,variable3);
```

A **user-defined SAS function** is **executed** in a method program, by a statement that has the following general form:

```
Variable-name = Function-name (parm1,parm2,parm3);
```

The parameters denoted by *parm1*, *parm2* and *parm3* are all input parameters. The variable specified in the *Return* statement, within the user-defined function, carries data values to the variable specified in *variable-name*.

Subroutines and functions with the same name may be stored in different packages. If a duplicate subroutine name exists, the subroutine name is prefixed with the package name upon usage. For example, in order to use the subroutine named *Linear* in the *Future* package, the subroutine name is specified as *Future.Linear* within the *Call statement.*

A special feature, named **variable suffixes** provides access to special portions or attributes of Risk Dimensions variables in method programs.

Some of the available suffixes are:

.Mat

The value of the Maturity attribute of a risk factor variable is accessed by this suffix.

.Dim

The number of elements in a risk factor array is accessed..

Vol

The volatility estimate of a risk factor variable is accessed by this suffix.

The data values contained in these suffixes are used pricing methods to execute the correct valuation of the instrument types. The use of the suffixes is illustrated in pricing methods of the *Casestudy_Env* in Section 7.4.3.

The six different kinds of method programs are subsequently discussed and illustrated in terms of the case study. A few closing remarks about the use of method programs are discussed in Section 7.4.6.

7.4.2 Instrument input methods

An **instrument input method** uses the data values of some instrument variables to derive new data values for other instrument variables. Consider the instrument variable *OptType* in the SAS data set *Tradebook.* The data values contained in this variable is either *EC*, *AC* or *EP.* This indicates a *European Call*, an *American Call* or a *European Put* option. The instrument input method created in Program Code 7.3 uses these data values to create new data values for the *Input_OptType* variable that are either *CALL* or *PUT*. The pricing method of the option instrument uses these values to perform the correct valuation.

Program Code 7.3: The Instrument Input Method *OptType_Input*

```
Proc  Compile  Env = "&RiskEnv" Outlib = "&RiskEnv" Package = Options;
Method OptType_Input kind = Input label = "Correct OptType";
If OptType = "EC" or "AC" then Input_OptType = "CALL";
If OptType = "EP" then Input_OptType = "PUT";
Endmethod;
Run;
```

7.4.3 Pricing methods

Each instrument type (see Section 7.5) must have an associated **pricing method**. The pricing method is used to calculate the value of each open position that is held in the instrument type. The calculated value is assigned to the system-defined variable, named _*Value*_. Output variables may be optionally assigned in pricing methods.

The pricing methods and theoretical formulas that are used to calculate each of the five financial instruments in the case study are subsequently discussed.

Options

The Black-Scholes formulas (cf. Hull(2003)) are used to value the European call (C) and put options (P). The formulas have the following form:

$$C = S_0 N(d_1) - Ke^{-rT} N(d_2) \tag{7.2}$$

$$P = Ke^{-rT} N(-d_2) - S_0 N(-d_1) \tag{7.3}$$

where

$$d_1 = \frac{\ln(\frac{S_0}{K}) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \quad \text{and}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

with

$N(x)$ = the cumulative probability distribution function for a standardized normal distribution,

S_0 = the current value of the underlying equity,

σ = the annualized volatility of the underlying equity,

T = the time to maturity of the option,

r = the continuously compounded risk-free rate of interest and

K = the strike price of the option.

It is important to note that an American Call option is valued by the same formula as a European call option. Program Code 7.5 is used to value the *Option* instrument type, making use of the formulas described above.

Program Code 7.5: The *Option_Prc* Pricing Method

```
Proc Compile  Env="&RiskEnv" Outlib="&RiskEnv" ;
Method Option_Prc kind=price;
Time = (Enddate - _date_)/365.25;
/*The subroutine RF_Interp is used to calculate the value of the risk-
free rate of interest for the corresponding time to maturity as
contained in the variable Time*/
Call RF_Interp(time,Zcurve.currency,Zcurve.currency.MAT,RF_Rate);
/*The Black-Scholes pricing formulas BLKSHCLPRC and BLKSHPTPRC are used
to value the Call and Put options. The variable suffix .Vol is also
used.*/
If Input_OptType = "CALL" then _VALUE_ =
BLKSHCLPRC(Strike,time,Price.Underlying,RF_Rate,Price.Underlying.Vol);
If Input_OptType = "PUT" then  _VALUE_ =
BLKSHPTPRC(strike,time,Price.Underlying,RF_Rate,Price.Underlying.Vol);
/*The difference between the value of the instrument type and the amount
initially paid for it, is contained in the output variable Daily_Profit
*/
Daily_profit = _VALUE_ - premium;
Endmethod;
Run;
```

Futures

The formula (cf. Hull(2003)) that is used to calculate the value of a future on an equity has the following form:

$$F_0 = (S_0 - C)e^{-rT} \qquad (7.4)$$

where

F_0 = The value of the future on the valuation date,

S_0 = the current value of the underlying equity,

C = fixed price payable on the maturity date for the equity,

T = the time to maturity of the future and

r = the continuously compounded risk-free rate of interest and

The pricing method that is used to value the *Future* instrument type is illustrated in Program Code 7.6.

Program Code 7.6: The *Future_Prc* Pricing Method

```
Proc Compile  Env = "&RiskEnv" Outlib = "&RiskEnv" ;
Method Future_Prc Kind = price;
Time= (Enddate - _date_)/365.25;
/*The subroutine RF_Interp is again used in the calculation of the
appropriate risk-free rate of interest. The variable suffix .Mat is also
used.*/
Call RF_Interp(Time,Zcurve.Currency,Zcurve.Currency.Mat,RF_Rate);
_VALUE_ = (Price.Underlying - ContractPrice)*(exp(-RF_Rate*time));
Daily_profit = _VALUE_;
Endmethod;
Run;
```

Equities

The *Equity_Prc* pricing method in Program Code 7.7 is used to calculate the value of the *Equity* instrument type. The value of equity is simply the latest observed market data value.

Program Code 7.7: The *Equity_Prc* Pricing Method

```
Proc Compile Env = "&RiskEnv" Outlib = "&RiskEnv" ;
Method Equity_Prc kind = price;
_Value_ = Price.MarketPrice;
Daily_profit = _VALUE_ - premium;
Endmethod;
Run;
```

Interest Rate Swaps and Government Bonds

The system-defined variable or structure, named *_Cashflow_* is used in the valuation of both interest rate swaps and government bonds. The characteristics of this variable are discussed before the theoretical formulas and pricing methods of the interest rate swaps and government bonds are discussed.

Consider the *_Cashflow_* variable. Elements that contain different values may be specified for this variable. The data values that are contained in the elements are used in the valuation process.

The available elements are:

.Num

The number of cash flows or payments that are used in the valuation of the instrument is stored in this element.

.

Matdate

The dates of the different cash flows are specified in this element.

.Matamt

It specifies the amounts payable on the cash flow dates.

.Repdate

The repricing dates of the cash flows are stored in this element.

.Repamt

It specifies the repricing amounts for the cash flows.

.CFint

The amount of interest income is stored in this element.

The first three elements are required. All the elements, except *.Num* may be either SAS arrays or scalars. The use of the *_Cashflow_* variable is illustrated in the pricing methods, discussed later in this section.

The formula that is used to calculate the value of an **interest rate swap** (V_{swap}) depends on the discounted value of the fixed (B_{fix}) and floating (B_{float}) payments.

The present value (cf. Hull(2003)) of the **fixed rate payments** is defined as:

$$B_{fix} = \sum_{i=1}^{n} ke^{-r_i t_i} + Le^{-r_n t_n} \tag{7.5}$$

where

k = the fixed payment made on each payment date,
t_i = time until the $(1 \le i \le n)$ payments are exchanged,
r_i = the risk-free rate of interest corresponding to maturity t_i and
L = the notional amount of money in the swap agreement.

The present value (cf. Hull(2003)) of the **floating rate payments** is defined as:

$$B_{float} = (L + k^*)e^{-r_1 t_1} \tag{7.6}$$

where

k^* = the floating rate payment made on the next payment date,
L = the notional amount of money in the swap agreement,
t_1 = time until the next payment is exchanged and
r_1 = the risk-free rate of interest corresponding to maturity t_1.

In a position where the case study company *Activegrowth* is to receive fixed payments and pay floating payments, the following equation (cf. Hull(2003)) holds:

$$V_{swap} = B_{fix} - B_{float} \tag{7.7}$$

If the type of payments were reversed, the signs before B_{fix} and B_{float} in (7.7) also reverse. Program Code 7.8 is used to value interest rate swaps, making use (7.5), (7.6) and (7.7).

Program Code 7.8: The *Swap_Prc* Pricing Method

```
Proc Compile  Env = "&RiskEnv" Outlib = "&RiskEnv";
Method Swap_Prc kind = price;
valdate  = _date_;
If (valdate > MaturityDate) then _VALUE_ = 0.0;
/*The number of remaining payments and the date of the first payment are
calculated*/
mondiff = Intck( 'month', valdate, MaturityDate);
nflow = int(mondiff/coupfreq) + 1;
firstdate = Intnx ('month', maturitydate, - (nflow - 1) * coupfreq)
            + day(maturitydate)-1;
/*The number of remaining fixed and floating payments are stored in the
.num suffix of the system-defined variable _Cashflow_*/
_CASHFLOW_.num       = nflow;
_CASHFLOW_.fix.num   = nflow;
_CASHFLOW_.float.num = nflow;
/*The payments that are received are either floating or fixed payments
*/
Fixsign = 1; flotsign = 1;
If RcveType = "Floating" then  fixsign = -1;
Else  flotsign = -1;
/*The dates of the remaining payments are determined and stored in the
.matdate suffix of the system-defined variable _Cashflow_*/
Do i = 1 to nflow;
 _CASHFLOW_.matdate[i] = Intnx("month",firstdate,(i-1)* Coupfreq)
                          + day(maturitydate)-1;
 _CASHFLOW_.fix.matdate[i]  = _CASHFLOW_.matdate[i];
 _CASHFLOW_.float.matdate[i] = _CASHFLOW_.matdate[i];
End;
/*The floating rate is determined by a lag time grid*/
Count = 0;
Do i = 1 to nflow;
 If (fromdate < valdate) and (count < 1) then
  Do;
   Count = 1;
```

Program Code 7.8 continues…

```
   ResetDate = Intnx('month', _CASHFLOW_.matdate[i],-1 *coupfreq) +
               day(maturitydate)-1;
   Time = (_CASHFLOW_.matdate[i] - valdate)/365.25;
   Floatrate = Rlag(floatingrate.ftr_name,(valdate-ResetDate)/365.25);
  End;
  Else
  Do;
  Time = (_CASHFLOW_.matdate[i] - valdate)/365.25;
  End;
/*The risk-free rate of interest is used to determine the discounted
value of the future payments*/
 Call RF_Interp(time, Zero_Curve, Zero_Curve.mat, Rfrate);
 _CASHFLOW_.fix.matamt[i]   = (0.5)*Fixrate*notional*fixsign*
                              exp(-1 *Rfrate*time);
 _CASHFLOW_.float.matamt[i] = (0.5)*floatrate*notional*flotsign*
                              exp(-1*Rfrate*time);
/*The notional amounts are added to the floating and fixed interest
payments*/
 If i = 1 then _CASHFLOW_.float.matamt[i] = _CASHFLOW_.float.matamt[i]+
                              flotsign * notional * exp(-1*Rfrate*time);
 If i = (nflow) then
 _CASHFLOW_.fix.matamt[i] = _CASHFLOW_.fix.matamt[i] +
                           notional*fixsign*exp(-1*Rfrate*time);
 If _CASHFLOW_.matdate[i] > Maturitydate then _CASHFLOW_.matamt[i]= 0;
End;
/*The present value of the fixed payments is calculated*/
FixSum = 0;
i = 1;
Do While (_CASHFLOW_.fix.matdate[i] <= Maturitydate and
          i <= _CASHFLOW_.fix.num);
          Fixsum = Fixsum + _CASHFLOW_.fix.matamt[i];
          i = i + 1;
End;
/*The present value of the next floating payment and notional amount due
are calculated */
Floatsum = 0;
Floatsum = _CASHFLOW_.float.matamt[1];
/*The value of the interest rate swap is the difference between the two
sums calculated above. */
_VALUE_ = Floatsum + Fixsum;
Daily_profit = _Value_;
Endmethod;
Run;
```

A relatively simple formula is used to calculate the value of **government bonds**. The formula is not the prescribed method used by the Bond Exchange of South Africa, as no allowance is made for accrued interest. The value of the bond is defined as the sum of the discounted future cash flows. An appropriate risk-free rate of interest is used for each future date. The value of the bond is formally defined as:

$$V_{bond} = \sum_{i=1}^{n} De^{-r_i t_i} + Ce^{-r_n t_n} \tag{7.8}$$

where

D = the periodical coupon payment,

C = the maturity amount payable on the maturity date,

t_i = time until the $(1 \le i \le n)$ payments are exchanged and

r_i = the risk-free rate of interest corresponding to maturity t_i.

The pricing method that is used to value government bonds in the *Casestudy_Env* is illustrated in Program Code 7.9. The method program uses (7.9).

Program Code 7.9: The *Gov_bond_Prc* Pricing Method

```
Proc Compile  Env = "&RiskEnv" Outlib = "&RiskEnv" ;
Method Gov_Bond_Prc desc = "Gov Bond Pricing method" Kind = price;
/*A very accurate method is used to determine the first coupon date
after the date of valuation*/
valdate = _date_;
valday = day(valdate);
valmonth = month(valdate);
valyear = year(valdate);
matday = day(maturitydate);
matmonth = month(maturitydate);
matyear = year(maturitydate);
mondiff = (matyear - valyear) * 12 + matmonth - valmonth;;
moninc = mod(mondiff, coupfreq);
If moninc = 0 and (valday > matday) then moninc = coupfreq;
nflow = (mondiff-moninc)/coupfreq + 1;
nextday = matday;
nextmon = mod(valmonth + moninc, 12);
nextyear = valyear + ( valmonth + moninc - nextmon )/12;
If nextmon eq 2 and nextday > 28 then do;
      If ( mod( nextyear, 4) eq 0 )
            then nextday = 29;
         else nextday =  28;
      end;
```

Program Code 7.9 continues…

```
If (nextmon eq 4 or nextmon eq 6 or nextmon eq 9 or nextmon eq 11)
      and nextday > 30 then nextday = 30;
Nextnum = nextday * 1000000 + nextmon * 10000 + nextyear;
Nextchar = put( nextnum, 8.);
Nextdate = input( nextchar, ddmmyy8.);
/*The SAS date value of the first coupon date is saved in the .matdate
suffix of the _Cashflow_ system-defined variable*/
_Cashflow_.matdate[1] = nextdate;
/*The dates of the remaining coupon payments are calculated*/
Do i = 2 to nflow;
 Lastmon  = month(_Cashflow_.matdate[i-1]);
 Lastyear = year(_Cashflow_.matdate[i-1]);
 nextday = matday;
 nextmon = mod(lastmon + coupfreq, 12);
 nextyear = lastyear + (lastmon + coupfreq - nextmon )/12;
 If nextmon eq 2 and nextday > 28 then
          Do;
          If ( mod( nextyear, 4) eq 0 )then nextday = 29;
          Else nextday =  28;
          End;
 If (nextmon eq 4 or nextmon eq 6 or nextmon eq 9 or nextmon eq 11)
   and nextday > 30 then nextday = 30;
 Nextnum = nextday * 1000000 + nextmon * 10000 + nextyear;
 Nextchar = put( nextnum, 8.);
 Nextdate = input( nextchar, ddmmyy8.);
 _Cashflow_.matdate[i] = nextdate;
End;
/*The value of the bond is calculated as the sum of the discounted
coupon and maturity payments*/
q = Coupfreq * 30/360;
prc = 0;
Do i = 1 to nflow;
 Time = intck( 'day', _date_, _cashflow_.matdate[i])/365.25;
 Call RF_Interp(time,Zcurve.currency,Zcurve.currency.MAT,RF_Rate);
 prc = prc + notional * coupon * q * exp ( - (RF_Rate * time) );
 If (i eq nflow) then
         Do;
            prc = prc + Red_Amount * exp ( - (RF_rate * time) );
         End;
End;
_VALUE_ = prc;
Daily_profit= _Value_ - Premium;
Endmethod;
Run;
```

7.4.4 Risk factor transformation methods

A **risk factor transformation method** is used to calculate data values for some risk factor variables from the market data values of other risk factor variables. A risk factor transformation method is used to calculate the data values of a yield curve in the case study. The yield curve consists of risk factor variables like *ZR_1_MTH*, *ZR_3_MTH*,…, *ZR_10_YEAR*. For example *ZR_1_MTH* refers to the zero rate that corresponds to a one month maturity value. The risk factor transformation method *Mod_Zerorates* is used to derive data values for these variables from the data values contained in the risk factor variables *UNSTD_1_MTH*, *UNSTD_3_MTH*,…, *UNSTD_10_YEAR*. The yield curve is used to calculate risk-free interest rates for certain specified maturities during the valuation of financial instruments like futures and options. Program Code 7.4 is used to create the risk factor transformation method *Mod_Zerorates* in the *Casestudy_Env* risk environment. It is important to note that the detail of Program Code 7.4 is only discussed in Chapter 9.

Program Code 7.4: The Risk Factor Transformation Method *Mod_ZeroRates*

```
Proc Compile  Env = "&RiskEnv" Outlib = "&RiskEnv" ;
Method Mod_ZeroRates Kind = trans;
%Unstandardize(UNSTD_1_MTH,1,ZR_1_MTH);
%Unstandardize(UNSTD_3_MTH,2,ZR_3_MTH);
%Unstandardize(UNSTD_6_MTH,3,ZR_6_MTH);
%Unstandardize(UNSTD_12_MTH,4,ZR_12_MTH);
%Unstandardize(UNSTD_18_MTH,5,ZR_18_MTH);
%Unstandardize(UNSTD_2_YEAR,6,ZR_2_YEAR);
%Unstandardize(UNSTD_30_MTH,7,ZR_30_MTH);
%Unstandardize(UNSTD_3_YEAR,8,ZR_3_YEAR);
%Unstandardize(UNSTD_42_MTH,9,ZR_42_MTH);
%Unstandardize(UNSTD_4_YEAR,10,ZR_4_YEAR);
%Unstandardize(UNSTD_54_MTH,11,ZR_54_MTH);
%Unstandardize(UNSTD_5_YEAR,12,ZR_5_YEAR);
%Unstandardize(UNSTD_66_MTH,13,ZR_66_MTH);
%Unstandardize(UNSTD_6_YEAR,14,ZR_6_YEAR);
%Unstandardize(UNSTD_78_MTH,15,ZR_78_MTH);
%Unstandardize(UNSTD_7_YEAR,16,ZR_7_YEAR);
%Unstandardize(UNSTD_90_MTH,17,ZR_90_MTH);
%Unstandardize(UNSTD_8_YEAR,18,ZR_8_YEAR);
%Unstandardize(UNSTD_102_MTH,19,ZR_102_MTH);
%Unstandardize(UNSTD_9_YEAR,20,ZR_9_YEAR);
%Unstandardize(UNSTD_114_MTH,21,ZR_114_MTH);
%Unstandardize(UNSTD_10_YEAR,22,ZR_10_YEAR);
Endmethod;
Run;
```

7.4.5 Other method programs

The last three kinds of method programs are discussed briefly in this section.

Postpricing method programs perform calculations based on the values that are created by the valuation of instruments. These methods are executed after the valuation is finished, but before the aggregation occurs.

Postvar methods are used to perform calculations based on the values created by the valuation of instruments and the statistics resulting from simulation analyses. The calculations are done after the instrument pricing and simulation analyses occur, but are before aggregation.

The last kind of method program called **project methods** are used in memory management. The programs are executed after the market states are created, but before any pricing is done.

These programs are advanced features of Risk Dimensions and are not discussed in further detail in this document.

7.4.6 Closing remarks

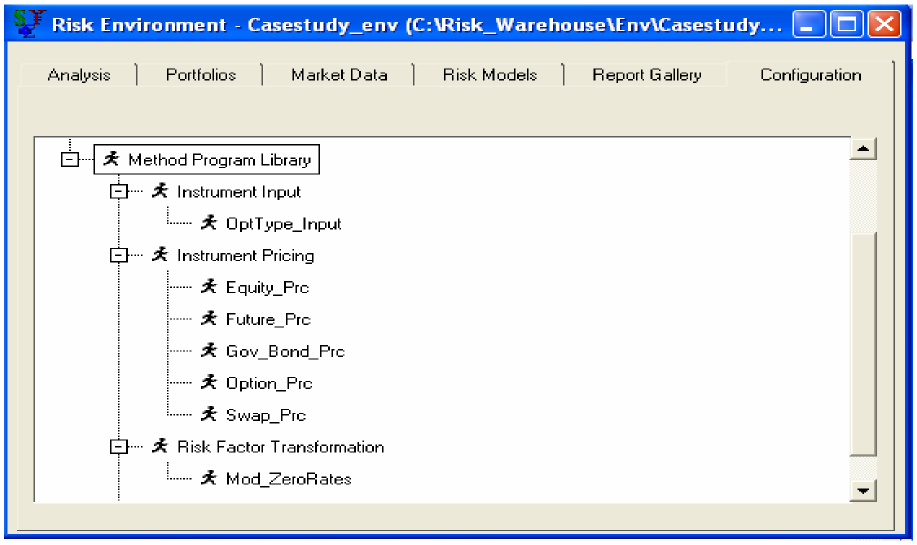
A last few remarks about method programs is made in this section.

The **variables** that are used in the program code of method programs must be either **registered** in the risk environment or first used in an **assignment** statement. The value of the undefined variable may be explicitly defined or it may be defined in terms of other variables. It also possible to include a large portion of the program code in subroutines or instrument input methods. This will make the length of the pricing methods shorter. The only problem is that the *_Cashflow_* system-defined variable may not be used in the subroutines and input methods. SAS arrays have to be used instead.

It is also important to note that pricing methods and the other kinds of method is are only created during this section and are **not executed**. The valuation of the financial instruments or instrument types in the portfolio is only done in Chapter 10.

Method programs may also be divided into **method blocks**. This enables time saving, as some parts of the program code in the method program are not run unnecessarily. This is very useful when handling large portfolios. The block-types that are used in Risk Dimensions are *Main*, *Init*, *Term*, *Inst_Init* and *Inst_Term*. Method blocks are also an advanced feature of Risk Dimensions and are not discussed further in this document.

The method programs that were created in the case study are viewed under the *Method Program Library* option in the *Configuration* tree of the GUI. The instrument input method with name *OptType_Input* is listed under the *Instrument Input* heading. The pricing methods *Equity_Prc, Future_Prc, Gov_Bond_Prc, Swap_Prc* and *Option_Prc* are listed under the *Instrument Pricing* heading. The risk factor transformation *Modzero_Rates* is listed under the *Risk Factor Transformation* heading. Figure 7.2 is used to illustrate the grouping of these structures in the GUI.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy...
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration

Method Program Library
- Instrument Input
  - OptType_Input
- Instrument Pricing
  - Equity_Prc
  - Future_Prc
  - Gov_Bond_Prc
  - Option_Prc
  - Swap_Prc
- Risk Factor Transformation
  - Mod_ZeroRates

Figure 7.2: The Method Program Library of the *Casestudy_Env*

7.5 Instrument types

In the financial world, companies and investors invest in the different financial instruments that are available. In the case study the company *ActiveGrowth* invested in options, equities, interest rate swaps, government bonds and futures. The next step in the risk management system, is to register each of these financial securities as a structure known as an **instrument type**. Each instrument type that is declared has an associated **pricing method** and **list** of Risk Dimensions **variables**. The pricing method is used to determine the value of each position held in this instrument. The data values of the list of variables that are specified, are necessary to value and classify the instrument. An **optional instrument input method** may also be assigned for an instrument type.

Instrument types may **inherit** attributes from other instrument types that have already been declared. Frequently used variables are then only defined once in a base type instrument and are inherited by other instrument types. The variables and pricing methods may also be inherited, if it is specified. The value of instrument types may be determined by discounted cash flows or by a quadratic approximating function. Only the discounted cash flows are discussed.

Instrument types are created by the ***Instrument*** statement. The following options may be specified in this statement:

Variables = (variable-list)

The variable list is required. All the variables that are used to value and classify the instrument are included in the list, except some of the system-defined variables, for example *InstType* (see Section 5.2).

Methods = (method-kind1 method-name1, method-kind2 method-name2)

One pricing method and a maximum of ten instrument input methods are assigned for each instrument type. The names specified in *method-name* option must already exist as method programs.

Valrecord = valuation-method

It specifies that instruments of this type include additional data to be used to value the instruments. The valuation-method option may be either specified as *Cashflows* or *Quadratic*.

Basetype = instrument-type-name

The attributes of instrument type assigned in the *instrument-type-name* option are used for the instrument type that is currently being declared.

Label = "Label-name"

A descriptive label may be specified.

Program Code 7.10 is used to create the necessary instrument types in *Casestudy_Env* risk environment. An instrument type is created for each of the following financial instruments: an equity, an option, a future, an interest rate swap and a government bond.

Program Code 7.10: The creation of Instrument Types in *Casestudy_Env*

```
Proc Risk;
Environment Open = "&RiskEnv";
/*The frequently used variables are defined in a base instrument type,
named Root.  No other instrument options are specified */
Instrument Root
           Variables = Holding,Book,Currency,ShortPosition,Premium);
/*An instrument type, named Option is created.  The instrument type by
name of Root is inherited with all its variables. Additional variables
are assigned in the Variables option.  The instrument input method,
named OptType_Input and the pricing method by name of Option_Prc are
used to value each position in an instrument of this type*/
```

Program Code 7.10 continues…

```
Instrument Option
           Basetype = Root
           Variables = (Strike, EndDate, OptType, Input_OptType,
                        Underlying)
           Methods =(Input OptType_Input, Price Option_Prc);
/*The instrument types named Equity, Future, Int_Swap, and Gov_Bond are
also created. The Root instrument type, with all its variables are often
inherited. A pricing method, as well as additional variables ,is
assigned for each instrument type. */
Instrument Equity
           Basetype= root
           Variables=(MarketPrice, Sector)
           Methods = (Price Equity_Prc);
Instrument Future
           Basetype = Root
           Variables = (Contractprice, Underlying,Enddate)
           Methods = (price Future_Prc);
Instrument Int_Swap
           Basetype = Root
           Variables = (Notional Maturitydate RcveType Fixrate
                        Ftr_name Coupfreq Currency Fromdate )
           Methods = (price Swap_Prc);
Instrument Gov_Bond
           Basetype = Root
           Variables = ( Maturitydate, Coupon, Notional, Coupfreq,
                         Red_Amount)
           Methods = (price  Gov_Bond_Prc);
/*The Casestudy_Env is saved, with the added instrument types declared
above*/
Environment save;
Run;
```

An unlimited variety of financial instruments may be registered as instrument types. The only restriction is that each instrument type must have an associated pricing method and list of variables.

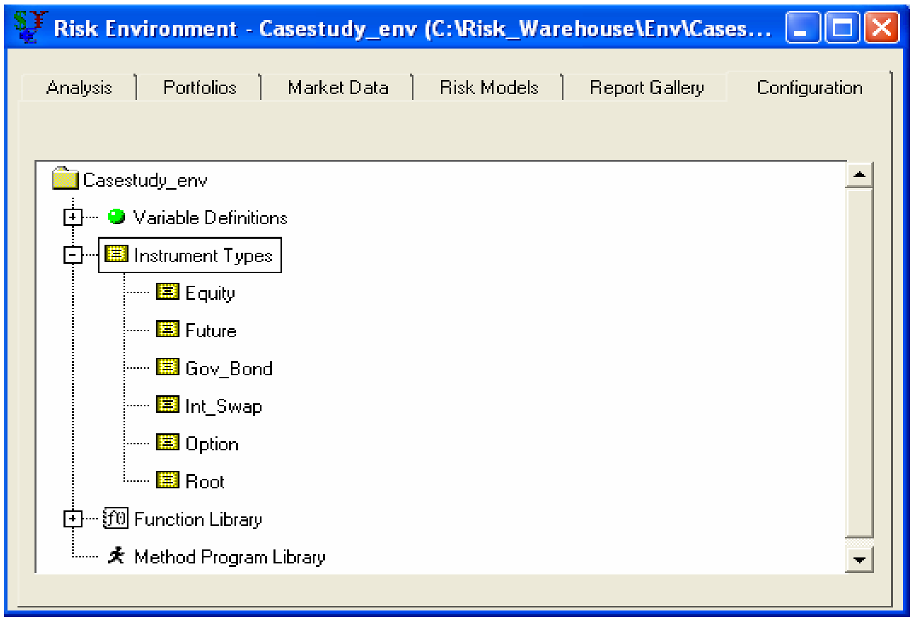
The system-defined variable *InstType* provides a link from the position data to the appropriate instrument type definition, for each instrument or record in the trade book or portfolio.

Consider the following partial record in the trade book:

InstType	**Instid**	**Short**	**Holding**	**Currency**	**Underlying**	**Enddate**	**Cprice.**
Future	SOL_QM4	1	4000	ZAR	SOL	17-Jun-04	97.86

The data value in the *InstType* variable is *Future*. In order to calculate the value of this position, an instrument type with the same name (*Future*) must be created, together with a pricing method and an appropriate variable list. If the data value contained in the *InstType* variable does not have a corresponding instrument type with the same name, an error occurs and the position is not valued.

The instrument types created in the *Casestudy_Env* risk environment are viewed under the *Instrument Types* heading in the *Configuration* tree of the GUI as illustrated in Figure 7.3.



Casestudy_env
- Variable Definitions
- Instrument Types
  - Equity
  - Future
  - Gov_Bond
  - Int_Swap
  - Option
  - Root
- Function Library
- Method Program Library

Figure 7.3 The Instrument Types in *Casestudy_Env*

7.6 Summary

Two Risk Dimensions structures, namely method programs and instrument types were created in this chapter. Method programs are used to access and change the data values of the Risk Dimensions variables that were registered in Chapter 5. Various method programs that differ in the type of variables that they access, are discussed in the chapter. The most important kinds are pricing methods, instrument input methods and risk factor transformation methods. An instrument type is created in Risk Dimensions to play a similar role to a financial instrument in the real world. A pricing method and a list of variables are assigned to each instrument type during registration.

The actual valuation of each position in the trade book only takes place in Chapter 10. The instrument types and trade book are used to create a Risk Dimensions structure named a portfolio file in Chapter 9.

8

RISK FACTOR MODELS

8.1 Introduction

Consider a risk factor variable, for example *ASA* that refers to the value of the ABSA equity price. The current value of this variable is used in the calculation of the mark-to-market (MtM) value of the portfolio. Suppose that a time series of historical daily closing values of this variable exists. A statistical model with a known form, for example the Garch(1,1) model may be fitted on the data values. The fitted model is used to **predict** the value of the risk factor variable, *ASA*, for a certain time in the future, for example the next trading day.

If a statistical model is fitted on the data values of a risk factor variable, it is called a **risk factor model**. A risk factor model may be fitted separately on each of the risk factor variables, or simultaneously on all the risk factor variables. The predicted future values of the risk factor variables are used together with Monte Carlo simulation to create an estimate of Value at Risk (VaR) for a certain time horizon. This method of calculating VaR is called model-based Monte Carlo simulation and is discussed in detail in Chapter 10.

The **general concepts** of statistical modeling are discussed in Section 8.2. The models that are used in the context of the case study are also discussed. The implementation of risk factor models in Risk Dimensions is discussed in detail in Section 8.3. The program code that is used to create the risk factor models in the

Casestudy_Env risk environment is discussed in Section 8.4. Some general remarks about an advanced statistical function, named copula function, are made in Section 8.5. The chapter ends with some closing remarks.

8.2 Statistical modelling

The key steps that are necessary in the process of fitting a statistical model are:

1. The formulation of the **model structure,**
2. the **distributional assumptions** that are made and
3. the **parameter estimation method** that is used.

The implementation of these steps is discussed, firstly, in terms of a simple statistical model in Section 8.2.1. Thereafter, each step is discussed in detail in Sections 8.2.2 to 8.2.4.

8.2.1 A simple statistical model

An introduction into the process of statistical modelling is discussed in this section. A simple statistical model, namely a straight line, is fitted on a set of observed data values. The knowledge that is gained during the introduction to statistical modelling will be useful when more complex models are considered later in the chapter.

Suppose a set of n observed data values for the variables x and y are available. Let (x_i, y_i) denote the i'th observed pair. A scatter plot may be used to illustrate the data values graphically. Suppose further that a statistical model namely a **straight line**, needs to be fitted on the observed data values. Then the fitted model is used to extrapolate from the observed data values.

The **model structure** is formulated as a mathematical equation. The structure of the model may, for example have the following form:

$$y_i = \alpha + \beta x_i + \varepsilon_i \qquad \text{for } i = 1,2,\ldots n \tag{8.1}$$

A few definitions about the symbols that comprise the formula are made. The variable y_i is defined as the **dependent** or **endogenous variable**, as it depends on the value of x_i. The variable x_i is defined as the **exogenous** or **independent variable**. The unknown constants α and β are called **model parameters**. The ε_i variable is defined as the **error term**.

The values of the variable $\{x_i\}$ may be used as input values in the equation and may be varied over a user-defined interval. The values of the parameters α and β may be estimated by a variety of parameter estimation methods. One method, namely the method of ordinary least squares, is discussed later in this section. The estimated parameters $\hat{\alpha}$ and $\hat{\beta}$ denote the intercept and slope of the line respectively. If the unknown parameters α and β are replaced by $\hat{\alpha}$ and $\hat{\beta}$ in (8.1) the model is said to be **fitted**. The **deterministic** or **non-random** part of the model structure may be defined as:

$$\hat{y}_i = \hat{\alpha} + \hat{\beta} x_i \tag{8.2}$$

where

$\hat{y}_i =$ the predicted value of the model corresponding to x_i.

The error term ε_i forms the **random** part of the equation. The error terms are calculated as the difference between the actual or observed value (y_i) and the predicted $(\hat{y}_i)$ value for each of the observed x_i values.

That is:

$$\hat{e}_i = y_i - \hat{y}_i \tag{8.3}$$

The fitted model may be used to extrapolate beyond the observed data values. The model, is thus, used to predict a value $\hat{y}_j$ for a value of x_j that does not form part of the observed data values $\{x_i, y_i\}$ that were used to fit the model. The error term ε_j and the actual value y_j is then unknown. It is useful to assume an identical **statistical distribution** for each of the error terms. Examples include: the normal, lognormal or the t-distribution.

The parameters α and β may be estimated by a variety of **parameter estimation methods**. One of these is the method of ordinary least squares. The function:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \alpha - \beta x_i)^2 \tag{8.4}$$

is minimized with respect to α and β. The estimates $\hat{\alpha}$ and $\hat{\beta}$ minimize the sum of the squared distances between the actual and predicted values of each value of y in the sample. The straight line that fits the data best, is thus obtained. The fitted statistical model, namely a straight line may be used in extrapolation.

8.2.2 The model structure

A more general model structure is considered in this section. The formulated model structure may consist of a **single equation** or a **system of equations**. The equations may further be, **nonlinear** in the parameters, nonlinear in the observed variables, or nonlinear in both the parameters and variables. The phrase “nonlinear in the parameters” means that the mathematical relationship between the variables and parameters is not required to have a linear form. The linear model as in (8.1) is a special case of a nonlinear model. The system of equations may be written as:

$$\mathbf{y} = f(\mathbf{y}, \mathbf{x}, \theta) + \boldsymbol{\varepsilon} \tag{8.5}$$

where

$\mathbf{y}$ = is a vector of endogenous or dependent variables,

$\mathbf{x}$ = is a vector of exogenous or independent variables,

$\boldsymbol{\theta}$ = is a vector of parameters and

$\boldsymbol{\varepsilon}$ = is a vector of unknown error terms.

Each equation in the system of equations in (8.5) defines a predicted value for a unique endogenous variable. It is important to note, that in the modelling of risk factor variables, the interest is sometimes not only in the mean of the endogenous variable, but also in the variance. Examples of more complex model structures are discussed in Section 8.2.5.

8.2.3 Distributional assumptions

The second step in the modeling process, is the distributional assumptions that are made with respect to the error terms $\{\varepsilon_i\}$. Normally the error terms are assumed to be independently, identically distributed. An assumption about the variance of $\{\varepsilon_i\}$is also made and tested. If the variance is constant, it is called homoscedasticity. If the variance is not constant, it is called heteroscedasticity. The distributional assumptions about the error terms also play a role in some of the parameter estimation methods, for example, maximum likelihood.

8.2.4. Parameter estimation methods

Consider the vector of unknown parameters $\boldsymbol{\theta}$ in the model equations. For example in (8.1) α and β are considered. A variety of **parameter estimation methods** may be used to estimate these parameters. The estimation methods share the following common goal: Find the set of parameters that make equation $f(\mathbf{y}, \mathbf{x}, \theta)$ predict well.

The following parameter estimation methods are generally used:

- Ordinary least squares (OLS),
- seemingly unrelated regression (SUR),
- generalized method of moments (GMM) and
- full information maximum likelihood (FIML).

The accuracy of these estimation methods depends on the data that are available.

8.2.5. Time dependent statistical models in the case study

A special type of statistical model, namely a dynamic or a **time dependent model**, is discussed in this section. The models that are used in the case study are discussed later.

Statistical models, where the current values of the endogenous variables depend on past or **lagged** values of the exogenous and endogenous variables, are used frequently in risk factor modelling. These types of models are called **dynamic** or **time dependent models**. Let $\hat{y}_t$ denote the predicted value of the endogenous variable at time t. The variable $\hat{y}_{t-i}$ is called the lag i of the variable $\hat{y}_t$. The Vasicek interest rate model and the Garch(1,1) model are examples of time dependent models. These models are used in the case study to model equity prices, volatilities and interest rates. They are subsequently discussed.

The **Vasicek interest rate model** has the following model structure:

$$rate_t = rate_{t-1} + \kappa \times (\theta - rate_{t-1})) + \varepsilon_t \ , \tag{8.6}$$

$$\mathrm{var}(rate_t) = \sigma \tag{8.7}$$

where

$rate_t$ = the predicted value of the interest rate at time t,

$\mathrm{var}(rate_t)$ = the variance of the interest rate,

κ, θ and σ are the unknown parameters and

ε_t = the error term at time t.

The unknown parameters are estimated and the value of $rate_t$ is predicted. The model is fitted on the risk factor variables *Prin1*, *Prin2* and *Prin3* in the next section.

A statistical model needs to be fitted on each of the **equities** that are used in the case study. The goal is to create a statistical model that accurately predicts future equity prices. The following notation definitions are made:

S_T = the price of the equity at time T,

S_0 = the price of the equity at time 0 and

μ = expected continuously compounded rate of return on the equity per annum.

Two assumptions about the behaviour of the equity prices are made, namely:

1. Expected return of the equity (μ) is constant and
2. The volatility of the equity price is zero.

If S_T grows at a continuously compounded rate of return per annum of μ, then (cf. Hull(2003)):

$$S_T = S_0 e^{\mu(T-0)} = S_0 e^{\mu T} \tag{8.8}$$

If time 0 is replaced by time $T-1$ in (8.8), it follows that:

$$\ln \frac{S_T}{S_{T-1}} = \mu \tag{8.9}$$

Consider the time series of closing values of each trading day that are available for each equity in the case study. Each time series of closing values is transformed by taking the logarithm of the ratio of the closing price to the previous day's closing price ($\ln \frac{S_T}{S_{T-1}}$). A statistical model is fitted separately on each of the transformed time series. The model is a straight line with a slope of 0 and an intercept parameter μ and forms part of the Garch(1,1) model that is formulated in (8.10). The equity prices in the case study that are fitted by this model in Section 8.4.2 are *ASA*, *AGL*, *SOL*, *SLM*, *ISC* and *OML*.

The **Garch(1,1)** model is used in the modelling of the **return** (see above) and **variance** of equity prices . The theoretical form is:

$$ret_t = \mu + \eta_t \tag{8.10}$$

$$\eta_t = \sqrt{h_t} \times \varepsilon_t \tag{8.11}$$

$$h_t = \varpi + \alpha \times \eta_{t-1}^2 + \beta \times h_{t-1} \tag{8.12}$$

where

ret_t = the predicted value of the log return of the equity price at time t,

h_t = the predicted variance of the log return of the equity price at time t,

ε_t = the error term at time t,

ϖ = the constant part of the volatility,

α = the coefficient of the lagged squared residuals and

β = the coefficient of the lagged volatility.

The Garch(1,1) model is fitted separately on the time series of the log ratio of the closing prices of each equity in Section 8.4. The square root of $\hat{h}_t$ is used as an estimate of the volatility of the equity prices. This estimate is used to predict values for *Vol_ASA*, *Vol_AGL*, *Vol_SLM*, *Vol_ISC*, *Vol_OML* and *Vol_SOL* in the case study.

8.3 Modelling in Risk Dimensions

8.3.1 The general structure of *Proc Model*

The SAS procedure named ***Proc Model*** is used to create statistical models in Risk Dimensions. The **statistical models** are fitted on the historic values of risk factor variables and are called **risk factor models**. The procedure is able to accommodate various user-defined model structures and provides a wide range of error distributions and parameter estimation methods. The statistical models that are created by this procedure are also registered directly as risk factor models in the appropriate risk environment in Risk Dimensions. The general structure of *Proc Model* is illustrated in Program Code 8.1.

Program Code 8.1: The general structure of *Proc Model*

```
Proc Model Data = Libref.SAS-data-set  Options;
Parm  p1  p2  …  pn ;
Endogenous y1 y2 … ym;
Model-Structure-Equations;
Optional statements;
Fit y1 y2 … ym  |  Fit options;
Run;
```

8.3.2 The options in the *Proc Model* statement

Several options are available in the ***Proc Model* statement** before the first semi-colon.

The name of the SAS data set that contains the historical data values that are used to estimate the parameter values in the model, is specified together with its SAS library in the ***Data*** option.

Several other **optional options** are available in the *Proc Model* statement. The ***Outspec*** option is used to save a **model specification** in a risk environment. A model specification is a model object that contains the equation(s) that define the structure of the model, but that are not fitted to a particular data set. The names and equations of the model specifications that are saved in a risk environment, are viewed in the *Risk Models* tree of the GUI. Although it is not required to create model specifications in the risk management system, it provides useful information about the model structures.

The available options in the ***Outspec*** option are:

Catalog

The name and SAS library of the SAS catalog that contains the appropriate risk environment, is specified in this option.

Modname = model-name

A suitable name for the model specification is specified in this option.

Modlabel = "Label"

A descriptive label is specified for the model specification.

Several other options exist in the *Proc Model* statement, but are not discussed.

8.3.3 The specification of the model structure in *Proc Model*

The **structure** of the risk factor model is formulated in the centre or main part of the *Proc Model* step. Equations are specified for the **mean** and **variance** of the risk factor variable(s). A variety of **lag functions** and **random number functions** may also be used in the equations that are formulated. A few examples of the program code that are used to formulate the model structures in *Proc Model* are included at the end of this section.

The model structure may be formulated either in **normal** or **general form**. Recall the model structure from (8.5):

$$\mathbf{y} = f(\mathbf{y}, \mathbf{x}, \theta) + \boldsymbol{\varepsilon} \qquad (8.5)$$

This formula is written in **normal form** as it is written in terms of an equation for the endogenous variable. It may also be written equivalently in **general form** as:

$$f(\mathbf{y}, \mathbf{x}, \theta) = \boldsymbol{\varepsilon} \qquad (8.12)$$

The error term is isolated one side of the equality sign in **general form**.

The model structure may be specified in normal or general from in the centre or main part of the *Proc Model* step. If the structure is specified in normal form it is automatically converted into general form during the execution of *Proc Model*. All the risk factor models that are formulated in this chapter are formulated in the normal form. The detail of the general form is omitted.

Consider (8.1). The following equivalent model structure is specified in *Proc Model*:

```
y = alpha + beta*x
```

Consider an endogenous variable *y* like in the equation above. The **prefixes** that follow may also be used in the model structure equations:

- *h.y* specifies the variance of *y*,
- *dert.y* defines the derivative $\frac{dy}{dt}$,
- *resid.y* is the residual of *y*,
- *nresid.y* is the normalized residual of *y*, calculated as $\frac{resid.y}{sqrt(h.y)}$,
- *MSE.y* is the mean-squared error value of *y*.

A variety of **lag functions** may also be used in the model structure equations. The three most commonly used functions are:

- *LagX*,
- *XLagX* and
- *ZLag*

The value specified by *LagX(i,variable)* function, is the i'th past value of the variable. The index *i* is not fixed and is bounded by *0* and *X*. Furthermore, if *i* is omitted, the X'th past value of the variable is used. If *X* is also omitted, the previous value of the variable is used. The *ZlagX(i,variable)* function is the same as the *LagX(i,variable)* function, except that all the missing values are set to zero. The *XlagX(variable,default)* function is also similar. The only difference is that the missing values are set to the value that is specified in the *default* option. The *XlagX* function is very useful in handling the start of a time series that is modelled by a time dependent model.

Advanced lag functions named *DifX*, *ZdifZ* and *MovavgX* are also available in the procedure, but are not discussed.

Random number functions may also be used in *Proc Model*. Examples include *Ranbin()*, *Rancau()*, *Ranexp()*, *Rannon()* and *Ranuni()*. The *Rangam()* function, draws, for instance a random number from a Gamma distribution.

Other SAS program code like the *If-then-Else* statement, the Do loop, as well as SAS arrays, may also be used in the formulation of the model structure equations.

Examples

Consider the **Vasicek interest rate model** in (8.6) and (8.7). The equivalent of these equations in *Proc Model* is written as:

```
Rate = Lag(rate) + kappa*(theta-Lag(rate));
h.Rate = sigma;
```

Consider the **Garch(1,1) model** in (8.10), (8.11) and (8.12). The equivalent of these equations in *Proc Model* is written as:

```
ret = mean;
h.ret = arch0 + arch1(resid.ret*resid.ret) + garch1*zlag(h.ret);
```

The program code of the Garch(1,1) and Vasicek models that are used in the case study, is discussed in more detail in Section 8.4.2.

8.3.4 Additional statements in the *Proc Model* step

Various other required and optional statements are available in *Proc Model*. The statements are:

Parameters

The names of the parameters that are used in the model are listed in this statement. Initial or starting values for the parameters may also be optionally defined.

Endogenous

The names of the dependent or endogenous variable(s) that are used in the model structure equations are included in this statement.

Exogenous

The names of the independent or exogenous variables that are used in the model structure equations are listed in this statement.

Errormodel

The distribution of the error terms in the model are specified in this statement. The empirical distribution may be specified.

ID

The name of the variable that is used to identify the lagged values of the endogenous and exogenous variables is specified in this statement. The data values of the variable that is specified, should be SAS date values, with a time interval between the values.

Restrict

Boundaries or restrictions on the values of the parameters in the model, may be specified in this statement.

Label

A suitable label may be specified for each parameter in the model.

These statements are used in the creation of the risk factor models that are used in the *Casestudy_Env* risk environment. The risk factor models are created in Section 8.4.2.

8.3.5 The *Fit* statement in *Proc Model*

The ***Fit*** statement in *Proc Model* is used to specify a **parameter estimation method** in the model, to **save** the fitted model in a risk environment, to control the **output** of the **fitted models** and to specify various **printing options**.

A **parameter estimation method** is specified in the *Fit* statement. The method that is specified is used to estimate the parameter values that make the model structure predict future values the best. The model structure with the estimated parameter values is called a **fitted model.** A fitted model may also be described as a model specification that was estimated on a particular data set.

Some of the parameter estimation methods that may be specified in the *Fit* statement are **ordinary least squares** (***OLS***), **seemingly unrelated regression**

(***SUR***), **generalized method of moments** (***GMM***) and **full information maximum likelihood** (***FIML***).

The first and sometimes the second derivatives of the specified model equations, have to be taken in order to estimate the parameter values. Analytical formulas are used to calculate the derivatives where possible. When this is not possible an **iterative minimization process** may be used. Risk Dimensions support two iterative processes, namely the **Gauss-Newton** method or the **Marquardt-Levenberg** method. The ***Method*** option in the *Fit* statement is used to specify either the ***Gauss*** or the ***Marquardt*** option. The maximum number of iterations that are used in the processes is specified by the value in the ***Maxiter*** option.

If the model is highly nonlinear, good **starting values** for the parameters have to be specified. One method is the specification of starting values for the parameters in the ***Parameter*** statement. A more sophisticated method entails the use of the ***Start*** and ***Startiter*** options in the *Fit* statement. The *Start* option is used to specify the starting values of the parameters. If more than one starting value is specified for one or more of the parameters, a grid search is performed over all the possible combinations of the values and the best combination is chosen as starting values for the iterations. The *Startiter* option is then used to specify the number of minimization iterations that are performed at each grid point. The use of these options leads to better parameter estimates for the model.

The ***Outcat*** option in the *Fit* statement is used to **save** a fitted model, together with, its results or output in a risk environment. The following options are available in this option:

Catalog

The name and SAS library of the SAS catalog that contains the risk environment, is specified in this option.

Modname = model-name

The name of the fitted model is specified in this option. This name appears in the risk environment.

Modlabel = “Label”

A descriptive label may be specified for the fitted model.

Dim

The dimension of the model is specified in this option.

Interval

The interval of the data is specified in this option.

Some examples of options that may be used in the *Fit* statement to control the **output** of the **fitted models** are:

Out = SAS-data-set

The name of the SAS data set that contains the residuals, actual values and predicted values of each fitted model, is specified. Only the residual values are printed by default.

Outactual

This option ensures that the actual values of the endogenous variable in the model are written to the SAS data set that was specified in the *Out* option.

Outpredict

If this option is specified, the predicted values of the fitted model are included in the SAS data set that is specified in the *Out* option.

Outest = SAS-data-set

The name of the SAS data set that contains the parameter estimates and optionally the covariances of the estimates, are specified in this option.

Outcov

This option ensures that the covariance matrix of the parameter estimates are written to the SAS data set specified in *Outest* in addition to the parameter estimates.

Various **printing options** are also available in the *Fit* statement. These options print information about the fitting of the risk factor model in the output window of the SAS window environment. These options are not discussed in this document, but may be useful in monitoring finer detail of the modelling process.

8.4 Risk factor models in the case study

Proc Model is used in Section 8.4.2 to fit risk factor models on the time series data of the risk factor variables in the case study. A structure named a **SAS macro** that is used in Section 8.4.2 is first discussed in Section 8.4.1. SAS macros are used to shorten the program code that is necessary to fit the risk factor models in Section 8.4.2.

8.4.1 SAS Macros

The use of macro variables were discussed in Chapter 2. **SAS Macros** are also used to eliminate the use of redundant program code. The difference is that a SAS macro may not only contain a character string, but may also contain a whole SAS procedure, like *Proc Model* for instance. Macros are used in the next section, to fit a risk factor model separately on each one of a series of risk factor variables, without duplicating program code.

The ***%Macro*** statement is used to create a SAS macro. The name of the macro, as well as input parameters, is specified in this statement. The macro is ended by the ***%Mend*** statement. The name of the macro object is also specified in this statement. A SAS procedure like *Proc Model* may be included between these statements. The macro is **activated** in program code by preceding the name of the macro with the ***%*** sign. Input parameters and output parameters may also be specified. The use of SAS macros is illustrated in Program Codes 8.2 and 8.3.

8.4.2 Case study

Consider the case study. The yield curve consists of risk-free interest rates with different maturities that were calculated from observed JIBAR rates and swap rates. The problem is that the interest rates are usually highly correlated. Principal components analyses were used to reduce the dimensions of the yield curve from twenty-two to three. Program Code 8.2 is used to fit a separate **Vasicek interest rate model** on the historical data of each of the three principal components, namely *Prin1*, *Prin2* and *Prin3*. A future value for each of the principal components is predicted. The predicted values of the three principal components are transformed into predicted values for the original twenty-two risk factor variables in the next chapter.

Program Code 8.2: The Vasicek model fitted on the principal components

```
/*A macro object named CreateYieldModel is created. It has the input
parameter named rate*/
%MACRO CreateYieldModel(rate);
/*The SAS data set Prindata in the SAS library Riskdata contains the
data values of the risk factor variables, used in the modelling process.
A model specification is created in the Outspec option.*/
Proc Model Data=Riskdata.Prindata
Outspec =(Env.Casestudy_Env Modname= PC_&Rate
          Modlabel="&Rate and Vasicek");
```

Program Code 8.2 continues…

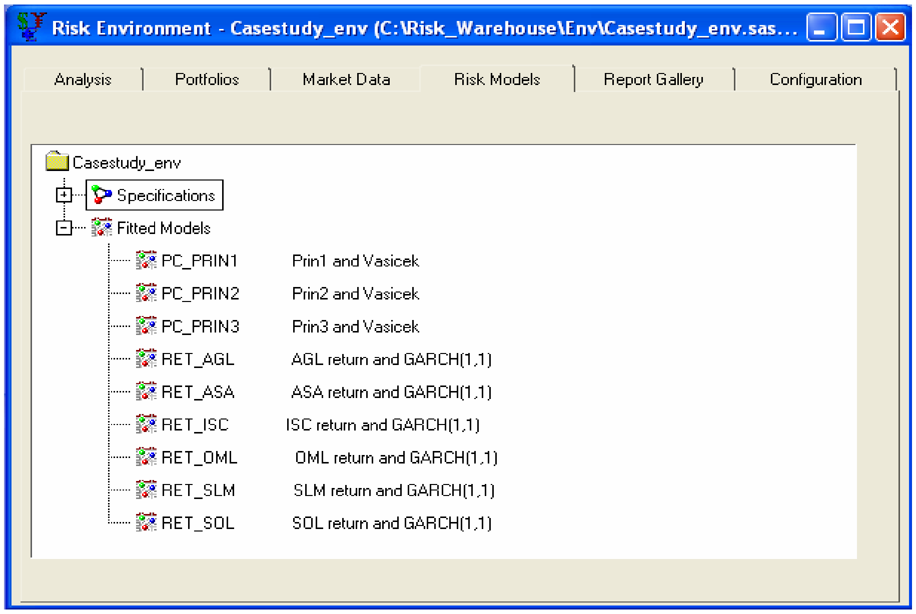
```
/*The endogenous variable, is the variable specified in the %Macro
statement*/
Endogenous &rate;
/*The names of the parameters are specified*/
Parameters kappa theta sigma;
/*The model structure equations are specified*/
&rate = Lag(&rate) + kappa * (theta - Lag(&rate)) ;
h.&rate  = sigma;
/*The Date variable is used to identify the lagged values of the
endogenous variables*/
Id date;
/*Descriptive labels are specified for the parameters*/
Label kappa = "Speed of Mean Reversion";
Label theta = "Long term Mean";
Label sigma = "Constant Variance";
/*The distribution of the error terms is assumed to be normal*/
Errormodel = normal;
/*The  name  of  the  endogenous  variable  that  is  fitted  is  specified.
Fitting options like the Method option, and the Fiml and Maxiter options
are used. Various output options, for example Outcov, Outest, Out and
Outpredict are also specified. The Outcat option is used to save the
fitted model in the risk environment.*/
Fit &rate / Method = Marquardt Fiml Maxiter = 100 Outcov Outactual
                     Outest=Models.cov&Rate Out = Models.res&Rate
                     Outpredict noprint
            Outcat =(Env.Casestudy_Env interval = weekday dim = 1
                     Modname=PC_&Rate Modlabel="&Rate and Vasicek" );
Quit;
/*The macro object is ended*/
%MEND createYieldModel;
/*The macro object is activated and is used to fit the Vasicek model on
each of the three principal components separately.*/
%CreateYieldModel(Prin1);
%CreateYieldModel(Prin2);
%CreateYieldModel(Prin3);
```

The **Garch(1,1)** model is used to predict future values of risk factor variables like *ASA*, as well as its corresponding volatility *ASA_Vol*, in the case study. Volatility is defined as the standard deviation of return or the square root of the variance of return. A SAS macro is used again and enables the use of one *Proc Model* procedure for six risk factor variables. Program Code 8.3 is used to create and save the the Garch(1,1) model in the *Casestudy_Env* risk environment.

Program Code 8.3: The Garch(1,1) model fitted on risk factor variables

```
/*The  %Macro  statement  is  used  to  create  the  macro  object  named
CreateModel  and  input  parameter  Eq.  The  statements  in  the  Proc  Model
procedure are broadly the same as in Program Code 8.2*/
%MACRO CreateModel(Eq);
/*The SAS data set Logreturns in the SAS library Riskdata contains the
historical data values of the logarithm of the daily return of the risk
factor variables.*/
Proc Model Data = Riskdata.Logreturns
Outparms = Models.parms_&Eq noprint
Outspec = (Env.Casestudy_Env Modname = ret_&Eq
           Modlabel = "&Eq return and GARCH(1,1)");
Endogenous  Ret_&Eq ;
Parameters  p0
            arch0
            arch1
            garch1;
Ret_&Eq = p0;
h.ret_&Eq = arch0 + arch1 * xlag(resid.ret_&Eq**2,mse.ret_&Eq)
             + garch1 * xlag(h.ret_&Eq,mse.ret_&Eq);
Restrict arch0 >= 0.00001,
         arch1 >= 0.00001,
         garch1 >= 0.00001,
         arch0 + arch1 + garch1 =1;
Label arch0  = "Constant part of conditional volatility";
Label arch1  = "Coefficient of lagged squared residuals";
Label garch1 = "Coefficient of lagged conditional volatility";
&Eq = zlag(&Eq)*exp(ret_&Eq);
Vol_&Eq =  sqrt(arch0 + arch1 * (xlag(ret_&Eq,resid.ret_&Eq) - p0)**2 +
           garch1 * zlag(Vol_&Eq));
Id date;
Errormodel = normal;
Fit Ret_&Eq / Method = Marquardt Fiml Maxiter = 1000 Outpredict
                       Outactual Outest = Models.cov&Eq
                       Out= Models.res&Eq
              Outcat = (Env.Casestudy_Env interval = weekday dim=1
             Modname = ret_&Eq Modlabel = "&Eq return and GARCH(1,1)" );
Quit;
%MEND CreateModel;
/*The macro object is used to fit the Garch(1,1) model on the six risk
factor variables*/
%CreateModel(ASA);
%CreateModel(OML);
%CreateModel(AGL);
%CreateModel(ISC);
%CreateModel(SLM);
%CreateModel(SOL);
```

The model specifications and the fitted models in the *Casestudy_Env* risk environment, are viewed in the *Risk Models* tree of the GUI. The model specifications are listed under the *Specifications* option and the fitted models under the *Fitted Models* option. Figure 8.1 is used to view the fitted models.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy_env.sas...
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration

Casestudy_env
Specifications
Fitted Models

PC_PRIN1	Prin1 and Vasicek
PC_PRIN2	Prin2 and Vasicek
PC_PRIN3	Prin3 and Vasicek
RET_AGL	AGL return and GARCH(1,1)
RET_ASA	ASA return and GARCH(1,1)
RET_ISC	ISC return and GARCH(1,1)
RET_OML	OML return and GARCH(1,1)
RET_SLM	SLM return and GARCH(1,1)
RET_SOL	SOL return and GARCH(1,1)

Figure 8.1 The *Risk Models* tree of the *Casestudy_Env* risk environment

8.5 Copulas

The use of the statistical function, named a **copula** is discussed briefly in this section.

Consider a set of risk factor variables that are necessary in the valuation of the portfolio. It is known that the use of a multivariable normal distribution for the set of risk factor variables is inappropriate. It is also known that it is not feasible to estimate all at once, a non-normal multivariate system of equations with hundreds of risk factor variables as dependent variables in a large parameter space. The alternative is that each risk factor variable is fitted separately with an appropriate distribution. The disadvantage of this method is that the use of separate models would ignore the interrelationships between the different risk factor variables.

The use of copula functions or simply, **copulas** in Risk Dimensions provides a solution to this problem. Separate risk factor models are again fitted on each of the risk factor variables. Copulas integrate the marginal distributions of the separate models into a single joint distribution for the set of risk factor variables. Copula methodologies is a study field on its own. It is an advanced topic in Risk Dimensions and is not discussed further in this document.

8.6 Summary

Consider the risk factor variables that are used in the valuation of the portfolio value of the case study company. Statistical models, called **risk factor models** were fitted on the historical data values of these risk factor variables, in this chapter. The models are used together with model-based Monte Carlo simulation in Chapter 10, to calculate an estimate of Value at Risk.

9

THE REGISTRATION OF MARKET AND PORTFOLIO DATA

Three topics are discussed in this chapter. The first topic is the use of **principal component analysis** in the case study. This process is discussed in detail in Section 9.1. The other two topics are the registration of **market** and **portfolio data** and are discussed in Section 9.2 and 9.3 respectively.

9.1 Case study: Principal Components Analysis

The use of **principal components analysis** in the case study, is discussed in this section.

Consider the yield curve that is used in the case study. The curve consists of **twenty-two** risk-free rates of interest or **zero rates**, each with a varying maturity value. The risk-free rates are calculated from observed market rates like the JIBAR rates and swap rates. A risk factor model (see Chapter 8) may be fitted on the historical data of each of the zero rates to obtain a predicted future value for each rate. The predicted values are used with Monte Carlo simulation in the calculation of Value at Risk as discussed in Chapter 10. This method of calculating an estimate of Value at Risk, is known as model-based Monte Carlo simulation.

The values in the twenty-two risk factor variables are usually **highly correlated** with each other. This presents a problem. The predicted values are each perturbed by either a negative or a positive value, leading to simulated values that are less correlated. The use of principal component analysis allows the use of a small subset of the risk factor variables, without losing too much information. The principal components are also less correlated with each other.

The steps that would be necessary to implement model-based Monte Carlo simulation firstly **without** and secondly **with** principal component analysis are listed below.

The necessary steps **without** PCA are:

1. Historical and current data values of the **original twenty-two zero rates** with varying maturities are calculated from swap and JIBAR rates and are stored in the *Yieldcurve_data* SAS data set (see Sections 3.2.1 and 3.4).

2. A **risk factor model** is fitted on the historical data values of each of the zero rates. A predicted future value for each zero rate is obtained.

3. **Monte Carlo simulation** (see Chapter 10) is used to perturb the predicted value of each zero rate. A series of simulated values is created for each zero rate.

4. The simulated zero rates and other simulated values risk factor variable values, are used together with portfolio information, pricing methods, instrument types and other Risk Dimensions structures (see Chapter 10) to calculate an estimate of **Value at Risk**.

PCA is used in the case study. The steps that are used in model-based Monte Carlo simulation are listed below. Some of these steps were done in previous chapters, some need to be done in this chapter and some are only done in Chapter 10.

The necessary steps are with PCA:

1. Historical and current data values of the **original twenty-two zero rates** with varying maturities are calculated from swap and JIBAR rates and are stored in the *Yieldcurve_data* SAS data set (see Sections 3.2.1 and 3.4).

2. **Principal components analysis** (see Section 6.4.3) is used to create three principal components from the original twenty-two zero rates.

3. A **risk factor model** is fitted on the historical data values of each of the principal components (see Section 8.4). A predicted future value for each principal component is obtained.

4. **Monte Carlo simulation** (see Chapter 10) is used to perturb the predicted value of each principal component. A series of simulated values is created for each zero rate.

5. The series of simulated values for each of the **three principal components** needs to be **reverted** into a series of simulated values for each of the **original twenty-two zero rates**. The steps that are necessary in the reversion, are discussed in detail in Sections 9.2.2 and 9.2.3. A theoretical discussion about the reversion follows later in this chapter.

6. The simulated zero rates and other simulated values risk factor variable values, are used together with portfolio information, pricing methods, instrument types and other Risk Dimensions structures (see Chapter 10) to calculate an estimate of **Value at Risk**.

The second process is subsequently discussed in more detail.

Step 1 was executed in Sections 3.2.1 and 3.4.

Consider **step 2**. The PCA was performed by the correlation method and the calculated value of the i'th principal component of the risk factor variables $\mathbf{X}' = [X_1, X_2, \ldots, X_{22}]$ is given by

$$\hat{Y}_i = \hat{\mathbf{e}}_{\mathbf{i}}'(\hat{V}^{1/2})^{-1}(\hat{\mathbf{X}} - \hat{\boldsymbol{\mu}}) \qquad \text{for} \qquad i = 1,2,...22. \qquad (9.1)$$

where

$\hat{\mathbf{e}}_i : (22 \times 1)$ = the eigenvector corresponding to the i'th largest eigenvalue $\hat{\lambda}_i$,

$\hat{\boldsymbol{\mu}}' = \overline{\mathbf{x}} = [\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_{22}]$

and

$$\hat{V}^{1/2} = \begin{bmatrix} \sqrt{s_{11}} & 0 & \cdots & 0 \\ 0 & \sqrt{s_{22}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{s_{22.22}} \end{bmatrix}$$

The principal components $\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_{22}$ were calculated by the SAS procedure *Proc Princomp* in Section 6.4.3. Only the first three largest principal components were used in further analyses, as they declares 98.616% of the total population variation.

The Vasicek risk factor model was used in Program Code 8.2 to predict the future value of each of the three principal components (*Prin1*, *Prin2* and *Prin3*) one trading day from the date of valuation (**step 3**).

Step 4 is only discussed in Chapter 10.

All the simulated values of the three principal components must be reverted to simulated values for the original twenty-two risk factor variables (**step 5**). A theoretical discussion about the transformation formula follows:

An equation for $\hat{\mathbf{X}}$ in terms of $\hat{\mathbf{Y}}$ must be derived. (9.1) is rewritten as (9.2) in vector and matrix notation.

$$\hat{\mathbf{Y}} = \hat{E}(\hat{V}^{1/2})^{-1}(\hat{\mathbf{X}} - \hat{\boldsymbol{\mu}}) \qquad (9.2)$$

where

$$\hat{\mathbf{Y}}' = [\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_{22}],$$

$$\hat{E} : (22 \times 22) = \begin{bmatrix} \hat{\mathbf{e}}_1' \\ \hat{\mathbf{e}}_2' \\ \vdots \\ \hat{\mathbf{e}}_{22}' \end{bmatrix}$$ is a matrix of eigenvectors and

the earlier definitions of $\hat{\mathbf{e}}_i$, $\hat{\boldsymbol{\mu}}$ and $\hat{V}^{1/2}$ that still apply.

Then, (9.2) is written in terms of $\hat{\mathbf{X}}' = [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_{22}]$ as follows:

$$\hat{\mathbf{X}} = \hat{\boldsymbol{\mu}} + \hat{V}^{1/2}\hat{E}'\hat{\mathbf{Y}} \qquad (9.3)$$

with

$\hat{E}'\hat{E} = I_{22}$ where I_{22} is the identity matrix of size 22×22.

If only the first three principal components are used (9.3) reduces to the following

$$\hat{\mathbf{X}} = \hat{\boldsymbol{\mu}} + \hat{V}^{1/2}\hat{E}'\hat{\mathbf{Y}} \tag{9.4}$$

where

$\hat{\mathbf{Y}} = [\hat{Y}_1, \hat{Y}_2, \hat{Y}_3]$ and

$\hat{E} : (3 \times 22) = \begin{bmatrix} \hat{\mathbf{e}}_1' \\ \hat{\mathbf{e}}_2' \\ \hat{\mathbf{e}}_3' \end{bmatrix}$ is a matrix of eigenvectors

The predicated and simulated values of $\hat{Y}_1, \hat{Y}_2$ and $\hat{Y}_3$ are contained in the risk factor variables *Prin1*, *Prin2* and *Prin3*. Three Risk Dimensions structures, namely a **linear transformation matrix** (see Section 9.2.2), **a risk factor transformation method** (see Section 7.4.4) and a **transformation set** (see Section 9.2.3) are used in the transformation process. The linear transformation matrix *Eigenvectors* in Program Code 9.3 is used to calculate the value of $\hat{E}'\hat{\mathbf{Y}}$. The risk factor transformation method *Mod_Zerorates* is then used to calculate the value of $\hat{\mathbf{X}}$ by multiplying $\hat{V}^{1/2}$ by $\hat{E}'\hat{\mathbf{Y}}$ and adding $\hat{\boldsymbol{\mu}}$. *Eigenvectors* and *Mod_Zerorates* are used in the transformation sets *Eigenvectortrans* and *Modzerotrans* respectively (see Section 9.2.3).

The vector $\hat{\mathbf{X}}' = [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_{22}]$ contains the simulated future values of the original twenty-two zero rates. The risk factor variables *ZR_1_MTH*, *ZR_3_MTH , …,ZR_10_YEAR* refer to the simulated zero rates.

The simulated values of the zero rates and other risk factor variables like *ASA*, together with portfolio information, pricing methods, instrument types and other Risk Dimensions structures are used in Chapter 10 to calculate an estimate of Value at Risk by the model-based Monte Carlo simulation methodology (**step 5**).

9.2 The registration of market data

The data values of risk factor variables that are observed in the market, are used in the valuation of instrument types and in the calculation of risk measures. The relevant market data values are stored in SAS data sets such as *Market_History* and *Yieldcurve_data* in the *Riskdata* library. The data values of these data sets must be made accessible for the registered pricing methods in the risk environment. A Risk Dimensions structure called a **market data source** is used for this purpose. It is a pointer to the actual location of the SAS data set and enables the access of the data values in Risk Dimensions and is discussed in Section 9.2.1. Another Risk Dimensions structure, called a **parameter matrix**, is discussed in Section 9.2.2. One form of parameter matrix, namely the **general type**, is created to contain market information that remains constant during the entire duration of the risk management system. The parameter matrix is linked to a SAS data set that contains the information which remains constant. The other type of parameter matrix, namely a **linear transformation matrix** is used in risk factor transformation. Risk factor transformation method programs (see Section 7.4.4) and linear transformation matrices are used in another Risk Dimensions structure, called a **transformation set**. The transformation sets are used to execute the risk factor variable transformations. The three Risk Dimensions structures are used in the registration of market data and are discussed in detail below.

9.2.1 Market data sources

As mentioned above, **market data sources** are used to make the market data in SAS data sets available to Risk Dimensions. They are pointers to the actual locations of the SAS data sets that contain the market data. Different types of market data sources exist. The type of risk analysis that is performed on the SAS data set determines the type of market data source that is created. The ***Marketdata*** statement is used in the *Proc Risk* procedure, to create market data sources. The general form of this statement is illustrated in Program Code 9.1.

Program Code 9.1: The general form of the *Marketdata* statement

```
Marketdata  Source-name
              File = “physical-location-file”
              Type = source-type
              Interval = interval
              Label = “label”;
```

The available **options** are:

Source-name

The **name** of the market data source that is created, is specified in this option.

File = “physical-location-file”

The **physical location** of the SAS data set that contains market data is specified in quotation marks, in this option.

Type = source-type

The type of the market data source, is specified in this option. The value of the type option may be one of the following: *Current*, *Timeseries*, *Covariance*, *Datafeed*, *Volatilities*, *Correlation*, *Changes*, *Scenarios*, *Transform* or *Parameter*.

The *Label* and *Interval* options are optional.

Different **risk analyses** require different **types** of market data sources. Table 9.1 contains examples of the risk analyses (see Chapter 10) that require certain market data sources.

Table 9.1: The required market data sources for certain risk analyses

Analysis	**Market data type**
All analyses	Current market data
Delta-Normal Analyses	Covariance matrix
Historical Simulation	Time series data
Scenario Simulation	Scenario data
Scenario, Profit/Loss curve and surface	Volatility data (Optional)
Monte Carlo Simulation	Covariance matrix (covariance-based simulation) or time series data (model-based simulation)

Program Code 9.2 is used to create eight market data sources in the *Casestudy_Env* risk environment.

Program Code 9.2: The creation of market data sources in *Casestudy_Env*

```
/*The name, type and interval of each market data source and the
physical location of each SAS data set is specified*/
Proc Risk;
Environment Open = "&RiskEnv";
            Marketdata Current File = "&RiskData\Current_Market"
                                   Type = current;
            Marketdata History File = "&RiskData\Market_history"
                                   Type =  timeseries
                                   Interval = weekday;
            Marketdata Yieldcurve_data
                                   File ="&RiskData\Yieldcurve_data"
                                   Type= timeseries
                                   Interval = weekday;
            Marketdata Princompdata
                                   File = "&RiskData\Prindata"
                                   Type = timeseries
                                   Interval = weekday;
            Marketdata Covar
                                   File = "&RiskData\Market_Covar"
                                   Type = covariance
                                   Interval = weekday;
```

Program Code 9.2 continues…

```
            Marketdata Eigenvectors
                                File= "&RiskData\Eigenvectors"
                                Type= transform
                                Interval = weekday;
            Marketdata Mean_Std
                                File= "&RiskData\Mean_Std"
                                Type= parameters
                                Interval = weekday;
            Marketdata Scenariodata
                                File="&RiskData\Scenariodata"
                                Type=scenarios;
Environment save;
Run;
```

9.2.2 Parameter matrices

It is not always necessary to assign Risk Dimensions variables to contain data values in the risk environment. If the data values that are used remain constant during the risk analysis process, it is easier to use a **parameter matrix**. A **general** type of parameter matrix exists, as well as a special type, called a **linear transformation matrix**.

The **general type** of parameter matrix is a Risk Dimensions structure that is used for the numerical values that stay constant and are used in both risk factor transformation method programs and pricing methods. The parameter matrix is first created from the SAS data set that holds the constant values. During the execution of the pricing methods and risk factor transformation methods, the values of the constants are then called from the parameter matrices. The difference between the use of risk factor variables and parameter matrices is that the values of risk factor variables may be changed during the analyses process. It is also necessary to register each risk factor variable, together with a range of variable attributes. It follows that it is easier to implement a parameter matrix for constant data values than it is to register a risk factor variable for each value separately.

The **general type** of parameter matrix is registered in a risk environment with the ***Parameter*** statement in *Proc Risk.* The following options available in this statement, for the registration of a general parameter matrix are:

Matrix-Name

A suitable name needs to be specified for the parameter matrix.

Data

The name of a market data source that already exists, is specified in this option. The market data source has to be of type *parameters.*

Column

The names of the columns of the SAS data set that are included in the parameter matrix are listed in this option. The order of listed names determines the order of the columns in the matrix. This option is not required. If it is left out, all the columns or variables except *_Name_* are read from the SAS data set.

Row

This option is used to identify and order the observations that are read from the SAS data set. The identifications are based on the data values of the *_Name_* column. This option is also optional and if it is left out, all the rows of the SAS data will be read into the parameter matrix.

The *Parameter* statement is used in Program Code 9.3 to create a general type of parameter matrix in the *Casestudy_Env* risk environment.

The **linear transformation matrix** is used in risk factor transformation. It may be used as an alternative or supplement to risk factor transformation methods. The matrix needs to be contained in a SAS data set.

Consider a linear transformation matrix $A:(n\times p)$. Further let:

$\mathbf{y}:(n\times 1)$ = a vector of input risk factor variables and
$\mathbf{z}:(p\times 1)$ = a vector of output risk factor variables.

The execution of the linear transformation matrix leads to the calculation of $\mathbf{z}$ where

$$\mathbf{z} = A'\mathbf{y} \tag{9.5}$$

A linear transformation matrix is used in the case study to transform the values of principal components into data values for the original risk factor variables.

The ***Parameter*** statement in *Proc Risk* is also used to create a **linear transformation matrix** in a risk environment. The following options are available in this statement:

Matrix-Name

A suitable name needs to be specified for the linear transformation matrix.

Data

The name of a market data source that already exists, is specified in this option. The market data source has to be of type *transform.*

Lintrans

This option is specified to ensure that the parameter matrix that is created by the *Parameter* statement is a linear transformation matrix.

Row

The names of the risk factor variables that contain the input values in the linear transformation matrix, are specified in this option.

Column

The names of the risk factor variables that receive the output data values, are specified in this option.

The linear transformation matrix, named *Eigenvectors* and the parameter matrix of general type named *Mean_Std*, is created in Program Code 9.3. *Eigenvectors* is used to calculate the value of $\hat{E}'\hat{\mathbf{Y}}$ in (9.4). Equivalently, it can be said that data values are calculated for the risk factor variables *UNSTD_1_MTH*, *UNSTD_3_MTH*,…, *UNSTD_10_YEAR* from the risk factor variables *Prin1*, *Prin2* and *Prin3.*

Program Code 9.3: The creation of parameter matrices in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
/*The name of the parameter matrix is Eigenvectors. A market data source
with the same name that is of type transform is specified in the Data
option. The Column, Row and additional Lintrans option is also
specified. Values are derived for Unstd_1_MTH, …, Unstd_10_Year from the
values contained in Prin1, Prin2 and Prin3.*/
Parameter  Eigenvectors
           Data=Eigenvectors Lintrans
           Column = (UNSTD_1_MTH  UNSTD_3_MTH UNSTD_6_MTH
            UNSTD_12_MTH UNSTD_18_MTH UNSTD_2_YEAR UNSTD_30_MTH
            UNSTD_3_YEAR UNSTD_42_MTH UNSTD_4_YEAR UNSTD_54_MTH
            UNSTD_5_YEAR UNSTD_66_MTH UNSTD_6_YEAR UNSTD_78_MTH
            UNSTD_7_YEAR UNSTD_90_MTH UNSTD_8_YEAR UNSTD_102_MTH
            UNSTD_9_YEAR UNSTD_114_MTH UNSTD_10_YEAR)
           Row = (Prin1 Prin2 Prin3);
/*The name Mean_Std is specified for the parameter matrix. A market data
source  with  the  same  name  (Mean_Std)  of  the  parameters  type,  is
specified  in  the  Data  option.  The  Column  and  Row  option  is  also
specified*/
Parameter Mean_Std
          Data=Mean_Std
          Column = (UNSTD_1_MTH  UNSTD_3_MTH UNSTD_6_MTH UNSTD_12_MTH
           UNSTD_18_MTH UNSTD_2_YEAR UNSTD_30_MTH UNSTD_3_YEAR
           UNSTD_42_MTH UNSTD_4_YEAR UNSTD_54_MTH UNSTD_5_YEAR
           UNSTD_66_MTH UNSTD_6_YEAR UNSTD_78_MTH UNSTD_7_YEAR
```

Program Code 9.3 continues…

```
           UNSTD_90_MTH UNSTD_8_YEAR  UNSTD_102_MTH UNSTD_9_YEAR
           UNSTD_114_MTH UNSTD_10_YEAR);
Environment save;
Run;
```

The values of a parameter matrix of **general type** may be retrieved in a **pricing method** or a **transformation** method, by using the ***PMXELEM*** subroutine. It is a built-in subroutine in Risk Dimensions and can return a single value or a full row or column from the matrix. The use of the *PMXELEM* subroutine is illustrated in Program Code 9.4.

Program Code 9.4: The *PMXELEM* subroutine in the case study

```
/*A macro object, named Unstandardize is created. Two input parameters
and an output parameter are specified. In the macro object data values
are retrieved from the parameter matrix, named Mean_Std. The retrieved
values are added to the value of the input parameter Unstdrate to form a
new value for the output parameter Zerorate.*/
%MACRO Unstandardize(Unstdrate,counter,Zerorate);
Call PMXELEM(Mean_Std,1,&counter,mean);
Call PMXELEM(Mean_Std,2,&counter,std);
&zerorate = mean + std * &unstdrate;
%MEND Unstandardize;
```

As previously mentioned, Program Code 9.3 was used to calculate data values for the twenty-two risk factor variables *UNSTD_1_MTH*, *UNSTD_3_MTH*,…, *UNSTD_10_YEAR* from the risk factor variables *Prin1*, *Prin2* and *Prin3.* The risk factor transformation method *Mod_Zerorates* in Program Code 7.4 uses these values, together with the SAS macro, named *Unstandardize* from Program Code 9.4 to derive values for the risk factor variables *ZR_1_MTH*, *ZR_3_MTH*,… ,*ZR_10_YEAR*. Consider (9.4) again:

$$\hat{\mathbf{X}} = \hat{\boldsymbol{\mu}} + \hat{V}^{1/2}\hat{E}'\hat{\mathbf{Y}} \qquad (9.4)$$

The *Mod_Zerorates* method uses $\hat{E}'\hat{\mathbf{Y}}$, multiplies it by $\hat{V}^{1/2}$ and adds $\hat{\boldsymbol{\mu}}$. This **concludes Step 5** of the second process, as simulated values for the original twenty-two zero rates are thus obtained. Program Code 7.4 is listed again.

Program Code 7.4: The Risk Factor Transformation Method *Mod_ZeroRates*

```
Proc Compile  Env = "&RiskEnv" Outlib = "&RiskEnv" ;
Method Mod_ZeroRates Kind = Trans;
%Unstandardize(UNSTD_1_MTH,1,ZR_1_MTH);
%Unstandardize(UNSTD_3_MTH,2,ZR_3_MTH);
%Unstandardize(UNSTD_6_MTH,3,ZR_6_MTH);
%Unstandardize(UNSTD_12_MTH,4,ZR_12_MTH);
%Unstandardize(UNSTD_18_MTH,5,ZR_18_MTH);
%Unstandardize(UNSTD_2_YEAR,6,ZR_2_YEAR);
%Unstandardize(UNSTD_30_MTH,7,ZR_30_MTH);
%Unstandardize(UNSTD_3_YEAR,8,ZR_3_YEAR);
%Unstandardize(UNSTD_42_MTH,9,ZR_42_MTH);
%Unstandardize(UNSTD_4_YEAR,10,ZR_4_YEAR);
%Unstandardize(UNSTD_54_MTH,11,ZR_54_MTH);
%Unstandardize(UNSTD_5_YEAR,12,ZR_5_YEAR);
%Unstandardize(UNSTD_66_MTH,13,ZR_66_MTH);
%Unstandardize(UNSTD_6_YEAR,14,ZR_6_YEAR);
%Unstandardize(UNSTD_78_MTH,15,ZR_78_MTH);
%Unstandardize(UNSTD_7_YEAR,16,ZR_7_YEAR);
%Unstandardize(UNSTD_90_MTH,17,ZR_90_MTH);
%Unstandardize(UNSTD_8_YEAR,18,ZR_8_YEAR);
%Unstandardize(UNSTD_102_MTH,19,ZR_102_MTH);
%Unstandardize(UNSTD_9_YEAR,20,ZR_9_YEAR);
%Unstandardize(UNSTD_114_MTH,21,ZR_114_MTH);
%Unstandardize(UNSTD_10_YEAR,22,ZR_10_YEAR);
Endmethod;
Run;
```

9.2.3 Transformation sets

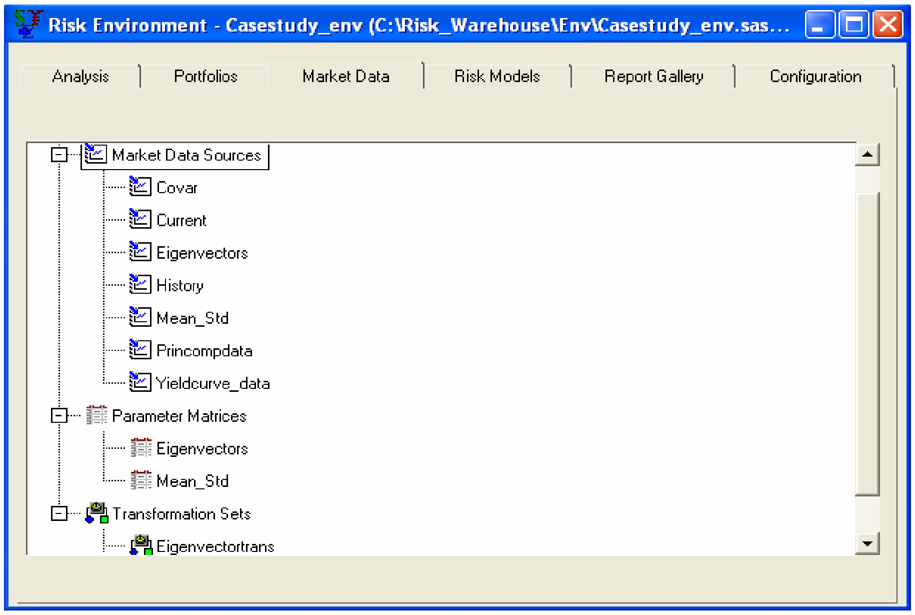
The Risk Dimensions structure, called a **transformation set** is used to execute risk factor transformation methods and linear transformation matrices. Only the transformation methods and transformation matrices that are defined as part of a transformation set are used in risk analyses. The ***Rftrans*** statement is used to specify the name of the transformation set, together with the risk factor transformation methods and linear transformation matrices that comprise the set.

An optional label may be specified. Program Code 9.5 is used to specify transformation sets in the *Casestudy_Env* risk environment.

Program Code 9.5: The creation of transformation sets in *Casestudy_Env*

```
/*A separate transformation set is created for the risk factor
transformation method Mod_Zerorates and the linear transformation matrix
Eigenvectors.*/
Proc Risk;
Environment open = "&RiskEnv";
Rftrans Eigenvectortrans Eigenvectors;
Rftrans Modzerotrans Mod_ZeroRates;
Environment save;
Run;
```

The contents of the *Market Data* tree in the GUI of the *Casestudy_Env* risk environment is viewed in Figure 9.1.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy_env.sas...
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration

Market Data Sources
- Covar
- Current
- Eigenvectors
- History
- Mean_Std
- Princompdata
- Yieldcurve_data

Parameter Matrices
- Eigenvectors
- Mean_Std

Transformation Sets
- Eigenvectortrans

Figure 9.1: The *Market Data* tree of *Casestudy_Env* in the GUI

9.3 The registration of portfolio data

In the financial world, a portfolio of financial instruments is kept by a financial institution. In our case, the case study institution named *Activegrowth*. The first step of implementing this portfolio in the SAS environment was to create raw data files that contain all the necessary information regarding open positions held. The next step was the conversion of the raw data files into SAS data sets. The information contained in the SAS data sets was used to register instrument variables that are used in pricing methods and instrument classification. The position information in the SAS data set needs to be accessed. The Risk Dimensions structure called a **portfolio data source** is created for each SAS data set that contains position information. It is a pointer to the physical location of the SAS data set. A **portfolio input list** is the next structure that is created. It is a list of portfolio data sources that is aggregated into one unified portfolio. The Risk Dimensions structure, named a **portfolio filter** is created next. This structure enables the creation of various portfolios by using conditioning logic statements on the data values of certain instrument variables. Lastly, a **portfolio file** is created by reading the data as specified in the portfolio input lists and portfolio filters. All the position information is now available in the risk environment and is ready for use by pricing methods.

9.3.1 Portfolio data sources

The information relating to the open positions held by a company are contained in SAS data sets such as *Tradebook, Swapbook* and *Bondbook*. In order to use the data values in these data sets in the risk environment, **portfolio data sources** are created. A portfolio data source is a structure that manages the way in which the position data is imported into the risk environment.

A portfolio data source is created by the ***Instdata*** statement in the *Proc Risk* procedure. The following options are available in this statement:

Name

A suitable name is specified for the portfolio data source.

File

The name and physical location of the SAS data set that contains the position information, is specified.

Format

The format of the data in the SAS data set is specified in this option. The available format options are *simple*, *sparse*, *cash flow* and *quadratic*.

Label

A descriptive label may be specified.

Variables = (Column-name = Instrument-Variable)

This option is used to rename the column names of the SAS data sets to instrument variables names. These instrument variables must already be registered in the risk environment. The rename option is specified in a *from = to* way, from the column name of the data set to the instrument variable name.

Type

If all the observations in the SAS data set are about the same instrument, then the *Type* option may be used to specify the name of the instrument. It is then, not necessary to include the system-defined variable *InstType* in the data set.

Here, we need to discuss the different formats in which the data values are stored in SAS data sets, as specified in the ***Format*** option, in more detail. A data set with a **simple** format contains one observation for each instrument. The

sparse format is used when the data set contains one observation for each field of each instrument. The **cash flow** format is used when each observation contains one cash flow of each instrument. The **quadratic** format is used for the data sets that contain the first and second derivatives of each instrument as observations.

Certain **system-defined variables** have to be included in the SAS data sets for specified **formats**. In the case study the **simple** format is used and only the requirements for these data sets are discussed.

A SAS data set with a **simple** format contains one observation for each instrument. The system-defined variables *InstID* and *InstType* have to be columns in the data set. The data values in *InstID* are unique and no duplication may exist. The variable *InstType* contains the names of the instrument types that were created in Section 7.4. The remaining columns of the data set contain the instrument variables that were declared in Section 5.2.3.

Program Code 9.6 is used to register the necessary portfolio data sources in the *Casestudy_Env* risk environment.

Program Code 9.6: The registration of portfolio data sources

```
Proc Risk;
Environment open = "&RiskEnv";
/*The Tradebook SAS data set in the SAS library Riskdata, is specified
in the File option.  A Label is specified, as well as, the simple format
in the Format option. The data set variable Shareprice is renamed to the
instrument variable Marketprice. The portfolio data sources Swapbook and
Bondbook are also registered.*/
      Instdata    Tradebook
                  File = "&RiskData\Tradebook"
                  Label = "Trade Book"
                  Variables = (Shareprice = Marketprice)
                  Format = simple;
      Instdata    Swapbook
                  File = "&RiskData\Swapbook"
                  Label = "Swap Book"
                  Format = simple;
```

Program Code 9.6 continues…

```
      Instdata    Bondbook
                  File = "&RiskData\Bondbook"
                  Label = "Bond Book"
                  Format = simple;
Environment save;
Run;
```

9.3.2 Portfolio Input Lists

A **portfolio input list** is a list of portfolio data sources. It is used to combine different portfolio data sources into a list that is used as input in the creation of a portfolio file. It is thus, an intermediate step in the creation of a portfolio file and is necessary even when only one portfolio data source exists. The ***Sources*** statement in *Proc Risk* is used to specify the name of the portfolio input list and the list of portfolio data sources that are included. Program Code 9.7 is used to combine the portfolio data sources *Tradebook*, *Bondbook*, and *Swapbook* into a portfolio input list with name *All_Deals_List*.

Program Code 9.7: The registration of portfolio input lists

```
Proc Risk;
Environment open = "&RiskEnv";
Sources All_Deals_List Tradebook Swapbook Bondbook;
Environment save;
Run;
```

9.3.3 Portfolio Filters

A **portfolio filter** is used to select only a **subset** of the information contained in a portfolio input list, to pass on to the portfolio file. A **logical expression** in program code is used to select the subset. A requirement is specified and only the observations (rows) that meet the requirement are used in the portfolio file.

The ***Filter*** statement in *Proc Risk* creates a portfolio filter. The *Where* option is used to create a conditional expression in the *Filter* statement. An optional label for the portfolio filter may also be specified. Only the names of registered instrument variables are used in the conditional expression. Portfolio filters with names *Derivatives_Filter, Commodity_Filter* and *Int_Der_Filter* are created in Program Code 9.8. For example, if *Derivatives_Filter* is used in the creation of a portfolio file, only the observations that have a data value of *Der* for the instrument variable *Book* will be included in the portfolio file.

Program Code 9.8: The registration of portfolio filters

```
Proc Risk;
Environment open = "&RiskEnv";
      Filter Derivatives_Filter where "Book='Der'";
      Filter Commodity_Filter where "Book='Com'";
      Filter Int_Der_Filter where "Book='Int_Der'";
Environment save;
Run;
```

The use of portfolio filters in the creation of portfolio files is discussed in the next section.

9.3.4 Portfolio Files

The final step in the creation of Risk Dimensions structures for position data, is the creation of **portfolio files**. A portfolio file is a binary SAS data file that is created in a format that is ready to be priced by the SAS Risk Engine. Thus, only portfolio files are used to represent position information in risk analyses.

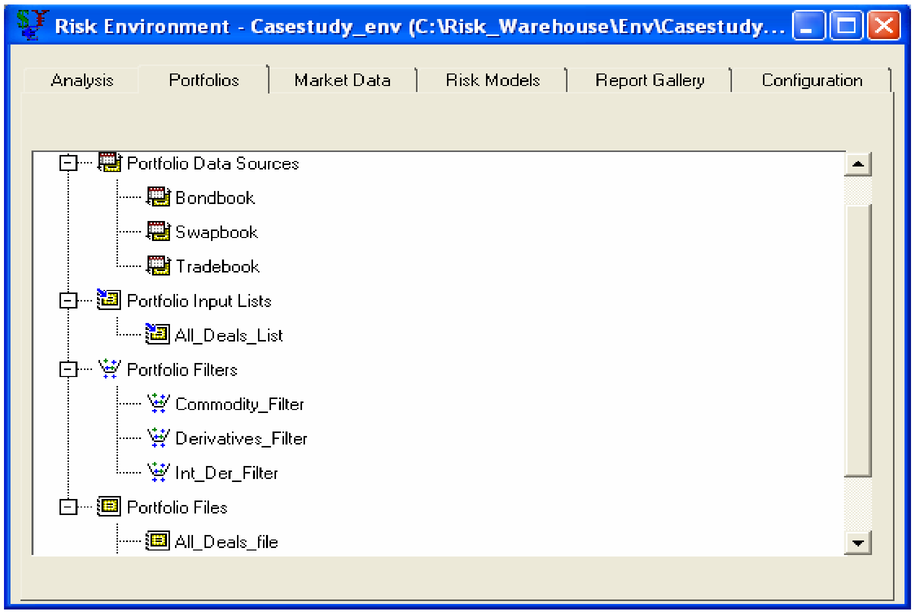
A **portfolio input list** and **optionally** a **portfolio filter** is specified as **inputs** to a portfolio files. A portfolio file is created by reading the data values contained in the SAS data sets, that are specified in the portfolio data sources, into Risk Dimensions. The portfolio data sources are specified in a portfolio input list.

The ***Read Sources*** statement in *Proc Risk* is used to create the portfolio files. A portfolio input list and an optional portfolio filter (in the *Filter* option) are specified in this statement. The name of the resulting portfolio file is specified in the ***Out*** option. More than one portfolio file may be created in a risk environment. One of the portfolio files created in Program Code 9.9 is a portfolio file by name of *Derivatives_File*. A portfolio input list named *All_Deals_List* and a portfolio filter named *Derivatives_Filter* is used in a *Read Sources* statement. In Section 7.5 a variables list was specified for each instrument type during registration. Only the variables specified then are included in the portfolio file. The portfolio file is static, meaning that if the position information of the company changes, a new file has to be created.

Program Code 9.9: The registration of portfolio files

```
Proc Risk;
Environment open = "&RiskEnv";
      Read Sources = All_Deals_List Out = All_Deals_file;
      Read Sources = All_Deals_List Out = Derivatives_File
                     Filter = Derivatives_Filter;
      Read Sources = All_Deals_List Out = Commodity_File
                     Filter=Commodity_Filter;
Environment save;
Run;
```

The contents of the *Portfolios* tree in the GUI of the *Casestudy_Env* risk environment is viewed in Figure 9.2.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy...
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration

Portfolio Data Sources
- Bondbook
- Swapbook
- Tradebook

Portfolio Input Lists
- All_Deals_List

Portfolio Filters
- Commodity_Filter
- Derivatives_Filter
- Int_Der_Filter

Portfolio Files
- All_Deals_file

Figure 9.2: The *Portfolios* tree of *Casestudy_Env*

9.4 Summary

The part that **principal component analysis** plays in the case study context was discussed in Section 9.1. The steps that are necessary for a successful implementation of PCA were discussed in detail. The need for Risk Dimensions structures such as parameter matrices and risk factor transformation methods were also illustrated in terms of PCA.

SAS data sets containing market information were registered in the risk environment, as **market data sources**. The role of other market data structures such as **parameter matrices** and **transformation sets** was also discussed in Section 9.2.

SAS data sets that contain position information are registered as **portfolio data sources** in a risk environment. The portfolio data sources may be combined into a **portfolio input list**. The list is used, together with a portfolio filter, to create a **portfolio file**. A portfolio filter may be used to create a portfolio file which is only a subset of all the open positions that are held by the company. The portfolio file is ready for use in pricing methods and risk analyses.

The market and portfolio data that were registered in this chapter are used in the next chapter, in the calculation of various risk measures.

10

RISK ANALYSES

10.1 Introduction

A wide range of analytical methods that may be used as risk analyses, exists in Risk Dimensions. The majority of the methods are grouped as either **market** or **credit** risk analyses. A third and smaller group, is defined as **general** risk analyses, as it may be applied to determine market or credit risk.

We first review the definitions of market and credit risk. **Market risk** is defined as the potential change in the portfolio value, due to a change in the values of the risk factor variables. **Credit risk** is defined as the potential change in the portfolio value, due to defaulting of obligations by counter parties. An example of credit risk is the potential defaulting of the counter party on the exchange payments of an interest rate swap.

The **market risk analyses** that are available in Risk Dimensions are discussed in detail in this **Section 10.2**. The analyses are:

- Sensitivity analysis
- Profit/Loss Curve analysis
- Profit/Loss surface analysis
- Scenario analysis and stress testing
- Delta-Normal analysis

- Simulation analyses
  - Historical simulation
  - Scenario simulation
  - Monte Carlo simulation

The execution of the market risk analyses, listed above, generates results that provide information about:

- How changes in risk factor variables values affect the portfolio value,
- the importance of risk factor variables relative to the portfolio value and
- an indication of the amount of risk that is contained in the portfolio.

The **credit risk analyses** that are available in Risk Dimensions are discussed in broad terms in **Section 10.3**. It is an advanced topic that falls outside the focus of this document.

Several **general risk analysis** methods are also available in Risk Dimensions. The methods may be used in both market and credit risk analyses. The available methods are discussed in **Section 10.4**.

A Risk Dimensions structure, called a **cross-classification** is discussed in **Section 10.5**. This structure is used to create **sub-portfolios** of the portfolio file. The results of the execution of risk analyses are reported separately for each sub-portfolio.

The market risk analyses that are created in Section 10.2 are executed in a Risk Dimensions structure, called a **Project**. This structure is discussed in **Section 10.6**. It is an important structure that brings together market data sources, cross-classifications, transformation sets, portfolio files and risk analyses. The projects are activated by the ***Runproject*** statement and for the first time in the implementation of the risk management system, the actual calculation of risk measures takes place.

The execution of a Risk Dimensions project leads to the creation of various SAS data sets. These data sets contain the results of the execution of the risk analyses and are called **output data sets**. Some of the most frequently used output data sets, as well as, the graphical illustrations that are also created by the execution of a project, are discussed in **Section 10.7**.

Two SAS statements, namely the ***Trace*** and ***%Include*** statements are discussed in **Section 10.8**. These statements are very useful in debugging and shortening SAS program code.

It is important to define the concept of market states at this stage. One data value is assigned to each risk factor variable for each trading day. The set of data values (one data value for each risk factor variable) is called a **market state**. The set of data values for the current date or date of valuation is called the **base-case market state**. A market state is necessary in the calculation of the portfolio value.

10.2 Market risk analyses

The creation of the different market risk analyses that are available in Risk Dimensions, is discussed in this section. These risk analyses are only executed in Section 10.6. The execution of each type of analysis brings different information into consideration for the risk manager.

10.2.1 Sensitivity analysis

Sensitivity analysis is used to provide information about how the portfolio value changes, if the value of a risk factor variable changes by a specified amount. The vector of first order derivatives (**deltas**) of the portfolio value with respect to a list of risk factor variables is calculated by default. It is also possible to calculate second order derivatives (**gammas**), **cross-derivatives**, derivatives with respect to time (**thetas**) and derivatives with respect to the volatility of risk factor variables (**vegas**). The calculations are all executed at the base case market state.

The ***Sensitivity*** statement in *Proc Risk* is used to create a sensitivity analysis structure in the risk environment. The statement has the following general form:

Sensitivity *Name Vars = Variable-list Options ;*

Various options are available in the *Sensitivity* statement. The following two options are required:

Name

The name of the sensitivity analysis structure is specified in this option.

Vars = variables-list

The names of the risk factor variables that are included in the sensitivity analysis, are specified. If this option is omitted derivatives are calculated with respect to all risk factor variables.

Several other options may also be specified in the *Sensitivity* statement in the ***Options*** option. They are:

Hessian

The use of this option specifies that the matrix of second and cross-derivatives is calculated.

Theta

This option requests that the derivatives with respect to time are also calculated.

Vega

The inclusion of this option enables the calculation of derivatives with respect to the volatilities of the specified risk factor variables.

Evaldate

The date on which the analysis is executed is specified in this option. If this option is omitted the date of valuation that is specified elsewhere is used.

Label = "Label"

A descriptive label may be specified for the sensitivity analysis structure.

Numerical or **analytical** techniques are available to calculate the derivatives. Only the default technique, namely the numerical method is discussed. The numerical formulas for delta (Δ) and gamma (Γ) follows:

$$\Delta = \frac{F(x+h) - F(x-h)}{2h} \tag{10.1}$$

$$\Gamma = \frac{F(x+h) - F(x) + F(x-h) - F(x)}{h^2} \tag{10.2}$$

Each one of the risk factor variables is perturbed up and down by the amount of h. The portfolio value is calculate as $F(x+h)$ and $F(x-h)$ for each of these movements.

A sensitivity analysis structure, named *Sensit* is created in the case study risk environment in Program Code 10.1. A list of risk factor variable names, as well as, the *theta* option is specified.

Program Code 10.1: Sensitivity analysis in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
Sensitivity Sensit Vars = (ASA AGL ISC SOL SLM OML Vol_ASA Vol_SOL)
                    theta;
Environment save;
Run;
```

Sensit is executed in Section 10.6. The results that are created by the execution of this structure are discussed in Section 10.7.

10.2.2 Profit/Loss curve analysis

The next risk analysis structure, a **Profit/Loss curve,** calculates the portfolio value for different market states that are created by varying a single risk factor variable over a grid of values, whilst the other variable values remains fixed. The portfolio value is recalculated at each point on the grid. The curve is a graphical structure that is used to visualize the impact that changes in the value of one risk factor can have on the portfolio value.

The ***PLcurve*** statement in *Proc Risk* is used to create Profit/Loss curves in the risk environment. Program Code 10.2 is used to illustrate the general form of this statement.

Program Code 10.2: The *PLcurve* statement

```
PLcurve      name
Curves= (rf_name_1 min =  min-value max = max_value n  =  num    type,
             rf_name_2 min =  min-value max = max_value n  =  num    type,
             rf_name_3 min =  min-value max = max_value n  =  num    type);
```

The ***name*** option in the *PLcurve* statement is used to assign a name to a set of Profit/Loss curves.

The specifications of each curve are listed in the ***Curves*** option and are separated from the next curve's specifications by a comma. Several options are available in the *Curves* option. They are:

rf_name_i

The name of the risk factor variable that is used in the i'th Profit/Loss curve, is specified in this option.

min

The starting point of the grid of risk factor variable values is specified in this option.

max

The end point of the grid of risk factor variable values is specified in this option.

n

The number of data values on the grid is specified in this option.

type

The type of perturbations that is used for the risk factor variable values, is specified in this option. The value specified in this option, the base case risk factor variable value and the values specified in the *min* and *max* options are used to determine the boundary points of the grid. The available options in *type* are:

- *Abs* The values in the *min* and *max* options are added to the base case value to create the boundary points of the grid.
- *Rel* The values in *min* and *max* are multiplied by the base case value to obtain the boundary points of the grid.
- *Std* The base case value is again multiplied by the values in the *min* and *max* options, but also by the standard deviation of the risk factor variable to obtain the boundary points of the grid.
- *Value* The specified values in the *min* and *max* options are used as the boundary points of the grid.

The value specified in the *n* option is used, together with the calculated starting and ending points of the grid, to determine the data values on the grid. Subsequent data values are a constant distance from each other.

Six Profit/Loss curves in two sets are created in the *Casestudy_Env* risk environment by Program Code 10.3.

Program Code 10.3: Profit/Loss curves in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
Plcurve Equity_Curve
      Curves = ( SOL min = 60 max = 85 n = 25 value,
                 ASA min = -0.20 max = 0.2 n = 21 rel,
                 SLM min = -3 max = 3 n = 21 abs,
                 ISC min = -0.50 max = 0.5 n = 25 std);
```

Program Code 10.3 continues…

```
Plcurve Rate_Curve
      Curves = (ZR_6_MTH min = -0.20 max = 0.2 n = 25 rel,
                ZR_2_YEAR min = -0.50 max = 0.5 n = 31 rel);
Environment save;
Run;
```

The sets of Profit/Loss curves, *Rate_Curve* and *Equity_Curve*, are executed in Section 10.6. The results that are created by the execution of these sets are discussed in Section 10.7.

10.2.3 Profit/Loss surface analysis

Profit/loss surface analysis is an extension of profit/loss curve analysis. The difference is that the portfolio value is calculated for market states that are created, by varying the values of two risk factor variables over a two-dimensional grid of values. The values of the rest of the risk factor variables are kept fixed. The portfolio value is recalculated at each point on the two-dimensional grid. The results of the analysis are illustrated in a graph.

The ***Plsurface*** statement is used to create a profit/loss surface. The general form of this statement is included in Program Code 10.4.

Program Code 10.4: The *PLsurface* statement

```
PLsurface     name
        rf_name1 (min= min-value max=max_value n = num    type)*
        rf_name2 (min= min-value max=max_value n = num    type),
        rf_name3 (min= min-value max=max_value n = num    type)*
        rf_name4 (min= min-value max=max_value n = num    type);
```

Program Code 10.4 illustrates the situation where one Profit/Loss surface analysis is created. The options that are used in the *PLsurface* statement are the same as the options in the *PLcurve* statement. Program Code 10.5 is used to create the profit/loss surface, named *Equity_Rate_Surface* in the *Casestudy_Env* risk environment. The values of the risk factor variables *ASA* and *ZR_6_MTH*, are perturbed.

Program Code 10.5: The *Equity_Rate_Surface* Profit/Loss surface

```
Proc Risk;
Environment open = "&RiskEnv";
PLsurface Equity_Rate_surface
       ASA( min = -0.20 max = 0.2 n = 21 rel)*
       ZR_6_MTH(min = -0.20 max = 0.2 n = 25 rel);
Environment save;
Run;
```

The *Equity_Rate_Surface* analysis structure is executed in Section 10.6. The results of the execution of this structure are discussed in Section 10.7.

10.2.4 Scenario analysis and stress testing

In **scenario analysis** and **stress testing** the value of the portfolio is calculated for user-defined values of one or more risk factor variables. The base case values are used for the remaining variables. The difference between the two analyses is that in **scenario analysis**, **relatively small changes** in the risk factor variable values are considered, whilst in **stress testing**, **extreme changes** in the risk factor variable values are considered.

Scenario analysis or stress testing is very useful when the investor calculates a projected value for a risk factor variable such as an interest rate and he or she wants to see the impact of this change on the portfolio value. The ***Scenario*** statement is used to create a scenario analysis or a stress testing structure. The general form of this statement is illustrated in Program Code 10.6.

Program Code 10.6: The *Scenario* statement

```
Scenario  name
Changes =  (rf_name1    value   type,
            rf_name2    value   type,
            rf_name3    value   type,
            rf_nameN    value   type)
Options ;
```

A suitable name is assigned to the scenario analysis or stress testing structure, in the ***name*** option.

The names of the risk factor variables, as well as the changes in their values, are specified in the ***Changes*** option. The following three options exist in this option:

rf_name1,…,*rf_nameN*

The names of the risk factor variables that are perturbed are specified in this option.

Value

Numerical values are specified in this option.

Type

This option may be specified as *Abs*, *Rel* or *Std.*

The values that are specified in the *Value* and *Type* options of the *Changes* option, determine the size of the changes in the risk factor variable values.

Additional options may also be specified in the *Scenario* statement in ***Options***. The following option is an example:

Kind

This option is used to specify the dependent variable that is analysed. The default value *PL*, is used to calculate the profit or loss arising from the scenario relative to the current portfolio value. The credit exposure of the portfolio under the scenario conditions are calculated by specifying the *Exposure* value in this option. The names of registered output variables may also be specified in this option.

Program Code 10.7 is used to create a scenario analysis and stress testing structure in the *Casestudy_Env* risk environment. A scenario analysis structure, named *Scenario1* and a stress testing structure, named *Stress1* are created. User-defined values for six risk factor variables are specified. These values, together with the base case values of remaining variables, are used to calculate the value of the portfolio. The portfolio value of the scenario is compared to the current portfolio value. Small changes to the base case values are specified in *Scenario1* and large changes in *Stress1*.

Program Code 10.7: Scenario analysis and stress testing in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
Scenario Scenario1
Changes = (ZR_3_MTH 0.005 abs, ZR_6_MTH 0.005 abs, SOL 0.1 rel,
           Vol_SOL 0.01 abs, ISC -0.1 rel);
Scenario Stress1
Changes = (ZR_3_MTH 0.05 abs,ZR_6_MTH 0.06 abs, SOL 0.55 rel,
           Vol_SOL 0.1 abs, ISC -0.5 rel);
Environment save;
Run;
```

The scenario analysis and stress testing structures that were created above, are executed in Section 10.6. The results that are created by the execution are discussed in Section 10.7.

10.2.5 Value at Risk (VaR)

A very important market risk measure, namely Value at Risk is discussed in this section. The calculation of Value at Risk is an attempt to quantify the total market risk in the portfolio into a single number or value. Consider a time horizon t and a confidence level p. **Value at Risk (VaR)** is defined as the loss in portfolio value over a time horizon t that are exceeded with probability $1-p$, (cf. Hull(2003). The definition may also be written into the following statement:

"The company is p percent certain that it will not lose more than VaR rand in the next t days."

An estimate of Value at Risk may be calculated by various methods. **Four different methods**, namely **Delta-normal analysis**, **historical simulation**, **covariance-based Monte Carlo simulation** and **model-based Monte Carlo simulation** are used in this chapter to provide estimates of VaR. Each one of the methodologies is discussed separately later in the chapter.

VaR is one the most informant and frequently used risk measurements in the financial world. Certain financial institutions such as large banks are also forced by legislation to calculate VaR as part of their risk management system. The calculation of VaR for the portfolio that is held by the company in the case study, is discussed later in this chapter.

10.2.6 Delta-Normal Analysis

Delta-normal analysis is used to calculate an estimate of the portfolio Value at Risk (VaR).

The analysis method is based on two **assumptions**. The first is that changes in the portfolio value have an approximately **linear** relationship with changes in the values of the risk factor variables. The second assumption is that the changes in the portfolio value, as well as, the changes in the values of the risk factor variables, are **normally** distributed.

The second assumption may be extended so that the changes in the risk factor variable values have a **multivariate normal distribution,** with a zero mean and a user-defined covariance matrix. The covariance matrix is estimated from the historical data values of the risk factor variables. The SAS procedure *Proc Corr* in Program Code 6.21 is used to perform the estimation. A theoretical discussion about covariance matrices is included in Section 6.4.2.

Suppose that the values of n risk factor variables are necessary in the calculation of the portfolio values. The formula that is used to calculate the Delta-Normal Value at Risk follows:

$$VaR_{DN} = z_\alpha \sqrt{\mathbf{X}'\hat{\Sigma}\,\mathbf{X}} \tag{10.3}$$

where

$\mathbf{X}' = [X_1, X_2, \ldots X_n]$,

$X_i = \Delta_i$ if the i’th risk factor variable has an interval measurement level,

$X_i = S_0\Delta_i$ if the i’th risk factor variable has a ratio measurement level,

Δ_i = the derivative of the portfolio value with respect to the i’th risk factor variable,

S_0 = the base case market state,

$\hat{\Sigma}$ = the user-provided covariance matrix and

z_α = the quantile of the standard normal distribution corresponding to a confidence level of α .

The ***Deltanormal*** statement in *Proc Risk* is used to create a Delta-Normal analysis structure in a risk environment. The following options are the most frequently used in the statement:

Data = covariance- market data-source

The market data source that contains the user-provided covariance matrix, is specified in this option.

Interval

The interval of the data in the covariance matrix is specified in this option.

Label

A descriptive label may be specified for the structure.

Conditional = category-list

This option specifies the risk factor variable category for conditional *VaR* analysis. Conditional VaR is calculated by perturbing all the values of the risk factor variables that belong in the *category-list*, while fixing the data values of the other risk factor variables.

Marginal= category-list

The risk factor variable categories for marginal VaR analysis are specified in this option. The values of the risk factor variables values in the *category-list* are fixed, whilst the values of the remaining variables are perturbed.

Program Code 10.8 which follows is used to create a Delta-Normal analysis structure, named *Delta_Sim* in the case study risk environment. An unconditional, a conditional and a marginal estimation of VaR are calculated. The *Conditional* option is used to perturb the values of the risk factor variables with *Commodity* specified in the *Category* attribute, whilst fixing the values of the other variables. The *Marginal* option is used to fix the values of the risk factor variables that have a *Volatility* category and to perturb the values of the remaining variables.

Program Code 10.8: Delta-Normal analysis in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
Deltanormal  Delta_Sim
      Data = Covar
      Interval = weekday
      Conditional = Commodity
      Marginal = Volatility;
Environment save;
Run;
```

The calculations of the Delta-Normal analysis, *Delta_Sim*, are executed in Section 10.6. The results that are obtained from the execution are discussed in Section 10.7.

10.2.7 Simulation analyses

Simulation analyses are used to create a set of potential market states. The portfolio value is calculated for each of the market states. This creates a distribution for the portfolio value, that is used in the calculation of Value at Risk and other risk measures.

Four simulation methods are available in Risk Dimensions. The difference between the methods is that they use different techniques in the generation of the potential market states. The methods are:

Historical simulation

The market states are based on historical risk factor variable values.

Scenario simulation

The market states are created by user-defined changes from the base case market state.

Covariance-based Monte Carlo simulation

The risk factor variable values are perturbed according to the user-provided covariance matrix, to generate market states.

Model-based Monte Carlo simulation

A statistical model is fitted to the historical data values of each of the risk factor variables. Future predicted values are obtained for each variable. The predicted values of the models are perturbed to generate market states.

The ***Simulation*** statement in *Proc Risk* is used to create a simulation analysis structure in a risk environment. The general form of this statement follows:

```
Simulation name Method = simulation-method   Options;
```

A suitable name for the simulation structure is specified in the ***name*** option. The simulation method that is used in the structure is specified in the ***Method*** option. The available methods are *Historical*, *MonteCarlo*, *Covariance* and *Scenario*. Each of the simulation methods have their own list of additional options that may be used in the *Simulation* statement.

The following optional options may, however, be specified for all the available methods:

Conditional = (category-list)

The list of risk factor categories that is used in the calculation of Conditional VaR, is specified in this option.

Marginal = (category-list)

The list of risk factor categories that is used in the calculation of Marginal Value at Risk, is specified in this option.

Horizon = (horizon-list)

A list of time horizons that are used in multiple horizon simulations is specified. The simulation analysis produces Value at Risk results for each time horizon point.

Kind = (variable-list)

This option specifies the dependent variables that are analysed in the simulation analysis. In addition to the default, *P/L,* any registered output variable may also be calculated.

Generator = generating-method

The method that is used to generate random numbers is specified, for example *pseudo*.

Ndraw = value

The number of simulation iterations is specified in this option.

Seed = value

The value specified in this option is used as starting value in the random number generation.

Each one of the four simulation methods are discussed separately in the remainder of the section.

Historical Simulation

Suppose that the closing values of all the risk factor variables for the past n trading days are available, then the daily changes in these historical values are used to create a set of possible market states for a specified time point (for example 1 trading day) in the future. The market states are used to create a distribution for the value of the portfolio. The $(1-\alpha)\%$ Value at Risk estimate is calculated as current portfolio value, minus the value of the $\alpha \times 100$'th percentile of the portfolio value distribution.

The ***Mlevel*** attribute is used to specify the measurement level of each risk factor variable as ***interval*** or ***ratio***. The set of possible future market states are created differently for interval and ratio risk factor variables. Let X_t be the value of a risk factor variable t trading days ago with X_0 the base case value. The following two methods are used in determining the market states:

Mlevel = Interval:

1. The changes are calculated as the **difference** between the consecutive values of the risk factor variable, i.e.

$$Change = X_t - X_{t-1} \quad \text{for } t = 1, \ldots, n-1$$

2. The new market states are created by **adding** the changes to the base case value of the risk factor variable, i.e.

$$Market\ state = X_0 + Change$$

Mlevel = Ratio:

1. The changes are calculated as the **ratio** of consecutive values of the risk factor variable, i.e.

$$Change = \frac{X_t}{X_{t-1}} \qquad \text{for } t = 1, \ldots, n-1$$

2. The new market states are created by **multiplying** the base case market value by the changes, i.e.

$$Market\ state = X_0 \times Change$$

One of the **strengths** of historical simulation is that no assumption is made about the underlying distribution of the changes in the values of the risk factor variables. Thus, it is also not necessary to estimate any parameter values. The methodology of historical simulation is easy to understand and consistent with the idea that the changes in risk factor variables are from any distribution.

The **weakness** of historical simulation is, however, the assumption that the set of possible future market states are fully represented by the changes in the historical values of the risk factor variables. Another way of stating this, is that all the possible future changes in the risk factor variable values for the specified time horizon have already occurred in the previous time period of n days. It is not always possible to get enough data on all the risk factor variables over a reasonable long period of time. The market states are then predicted on a limited amount of data and historical simulation may lead to an unreliable estimate of VaR.

Four additional options are available in the ***Simulation*** statement for historical simulation. The options are:

Data

The name of the market data source that is of the *time series* data type is specified.

Startdate

The date of the first observation from the historical data that is used in the simulation analysis, is specified.

Enddate

The date of the last observation from the historical data that is used in the simulation analysis, is specified.

Overlap/Nooverlap

This option is used if the time horizon of the simulation is larger than 1, for example 3. The market states are then generated by using the changes resulting from three consecutive observations of historical data (for example observations 1, 2 and 3). If the *Overlap* option is specified, the next market state is generated by observations 2, 3 and 4. However, if the *Nooverlap* option was specified, the next market states would have been generated by observations 4, 5 and 6.

Program Code 10.9 is used to create the historical simulation structure, named *Hist_Sim* in the *Casestudy_Env* risk environment. The method of simulation is chosen as *Historical*, the market data source *History* is specified, the interval of the data is chosen as *Weekday* and suitable starting and ending dates for the simulation are specified.

Program Code 10.9: The historical simulation structure *Hist_Sim*

```
Proc Risk;
Environment open = "&RiskEnv";
Simulation Hist_Sim Method = Historical
                    Data= History
                    Interval = weekday
                    Startdate = '05NOV2003'd
                    Enddate = '13MAY2004'd;
Environment save;
Run;
```

The execution of the historical simulation structure named *Hist_Sim* is discussed in Section 10.6. Various output data sets and a graphical illustration is created by the execution and is discussed in Section 10.7.

Scenario Simulation

Similar to historical simulation, **scenario simulation** is used in Risk Dimensions to create a list of market states, by adding changes in the risk factor variable values to the base case values. The difference between the two simulation methods is that in scenario simulation the market states are based on **user-defined** changes in the risk factor variable values. A SAS data set that contains a list of user-defined values for each of the risk factor variables is created. The changes in the consecutive risk factor variable values that are read from the data set are added to the base case values to create a list of market states. The value of the portfolio is calculated for each market state, creating a distribution for the portfolio value.

Scenario simulation may be very effective if the values in the SAS data set are based on good investor information. If the company believes that the scenarios created are likely to happen in the foreseen future it may adjust the make-up of the current portfolio.

Two additional options in the ***Simulation*** statement are used in scenario analysis. The options are:

Data

The name of a market data source that is of the *scenarios* data type is specified.

Match

This option is used to match values from the scenario data set. The matching may be done by date *(Match = Date)* or by horizon *(Match = Horizon)*. If no matching is done the *(Match = None)* option is chosen.

Consider the case study. The SAS data set named *Scenariodata* contains the list of user-defined values of each of the sixteen risk factor variables that are used in the valuation of the portfolio. The data set was registered as a market data source with the same name in Program Code 9.2. A scenario simulation structure, named *Scen_Sim* is created in the *Casestudy_Env* risk environment by Program Code 10.10. The method is specified as *Scenario*, the market data source named *Scenariodata* is specified as well and no matching is taking place.

Program Code 10.10: The scenario simulation structure *Scen_Sim*

```
Proc Risk;
Environment open = "&RiskEnv";
Simulation Scen_Sim Method = Scenario
                    Data = Scenariodata;
Environment save;
Run;
```

The execution of the *Scen_Sim* is discussed in Section 10.6. The output data sets and graphical results that are created by the execution, is discussed in Section 10.7.

Monte Carlo simulation

Monte Carlo simulation is used to good effect, if the assumption that the changes in the risk factor variables values have a certain underlying statistical distribution, is made. Monte Carlo simulation may be used to simulate a set of market states from any statistical distribution. The portfolio is again valued for each market state, creating a distribution of portfolio values. An estimate of VaR is calculated from the distribution.

The reliability of the distribution assumptions and the accurate estimation of parameter values are central to the effectiveness of Monte Carlo simulation and thus, the reliability of the Value at Risk estimate calculated by this method.

Monte Carlo simulation also uses historical values of the risk factor variables to generate possible future scenarios. The advantage of this method over historical simulation is, however, that the set of predictions are not limited to the exact scenarios that were observed in the historical data set. Another advantage of Monte Carlo simulation is that it is not constrained by normality as applies to the Delta-normal method.

The two available Monte Carlo simulation methods, namely covariance-based and model-based simulation are subsequently discussed.

Covariance-based Monte Carlo simulation

Covariance-based Monte Carlo simulation is used to provide another method of calculating a Value at Risk estimate. This method produces a set of market states by **assuming** that the changes in risk factor variable values are normally distributed.

The same user-provided covariance matrix that is used in Delta-Normal analysis, is also used in this method. The matrix is estimated from the historical data values of the risk factor variables and is of size $p \times p$.

The first step in this method is the calculation of the Cholesky root, L, of the user-provided variance-covariance matrix, Σ. The formula is:

$$\Sigma = LL' \tag{10.4}$$

The Cholesky root matrix is a lower triangular matrix of size $p \times p$ and is the square root of the covariance matrix.

A vector $\boldsymbol{\varepsilon}$ of p independent identically distributed $N(0,1)$ random variables is subsequently created. It is known that

$$Var(\boldsymbol{\varepsilon}) = E(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}') = I_p \tag{10.5}$$

A set of correlated perturbations, $\boldsymbol{\eta}$, is created by multiplying L by $\boldsymbol{\varepsilon}$:

$$\boldsymbol{\eta} = L\boldsymbol{\varepsilon} \tag{10.6}$$

It follows that:

$$V(\boldsymbol{\eta}) = V(L\boldsymbol{\varepsilon}) = L(V(\boldsymbol{\varepsilon}))L' = LI_pL' = \Sigma \tag{10.7}$$

Hence, the correlated perturbations have the same variability and correlation as the original risk factor variable data values. The next step in the method is to add the correlated perturbations to the base case market state, to create a simulated market state.

If a risk factor variable is defined as having an **interval measurement level**, the perturbations are added to the base case. For the i'th risk factor variable:

$$X_1 = X_0 + \eta_i$$

However, if the risk factor variable is defined as having a **ratio measurement level**, the following formula is used for the i'th risk factor variable:

$$X_1 = \exp(\eta_i + \ln(X_0))$$

The process is to create a new vector $\boldsymbol{\varepsilon}$, leading to a new vector $\boldsymbol{\eta}$ and a new market state is repeated for a number of iterations. The portfolio value is calculated for each of the market states. The distribution of portfolio values is used to obtain an estimate of Value at Risk.

Program Code 10.11 is used to create a covariance-based Monte Carlo simulation structure, named *Cov_Sim* in the *Casestudy_Env* risk environment. The simulation method is specified as *Covariance*, the data interval as *weekday* and the predicted time horizon as *1* day. The *seed*, *ndraw* and *generator* options are also specified.

Program Code 10.11: The covariance-based simulation structure *Cov_Sim*

```
Proc Risk;
Environment open = "&RiskEnv";
Simulation Cov_Sim Method = Covariance
                   Interval = weekday
                   Seed = 54321
                   Ndraw = 1000
                   Generator = pseudo
                   Horizon =1;
Environment save;
Run;
```

The execution of *Cov_Sim* is discussed in Section 10.6 and the results that are obtained from the execution are discussed in Section 10.7.

Model-based Monte Carlo simulation

Risk factor models were discussed in chapter 8. An equation structure was specified for each risk factor variable and the required model parameters were estimated. The models were used to forecast future values of the risk factor variables. The fitted models are perturbed in this section by **model-based simulation** to create a set of random market states. The portfolio value is calculated again for each market state. The resulting distribution of the portfolio value is used to calculate an estimate of VaR.

The following additional options in the ***Simulation*** statement are frequently used in Model-based Monte Carlo simulation.

Data

The name of the market data source that is used to fit the risk factor models, is specified in this option.

Errmod

The error distribution that is used in Monte Carlo simulation is specified. It may be the distribution that was used to fit the model (*asfit),* the empirical distribution (*empirical*) with t-distribution tails or the normal distribution (*normal*).

Interval

The data interval that is used in simulation is specified.

Consider the case study again. A risk factor model is fitted to fifteen risk factor variables of the sixteen risk factor variables, that are used to calculate the value of the portfolio. The risk factor variable *JB_6_MTH* is used to refer to a floating rate in interest rate swaps and is not modeled. Program Code 10.12 is used to create the model-based Monte Carlo simulation structure, named *Model_Sim* in the *Casestudy_Env* risk environment. The *Montecarlo* method and the market data source *History* are specified. The *interval*, *Errmod, seed*, *ndraw*, *generator* and *horizon* options are also specified.

Program Code 10.12: The model-based simulation structure *Model_Sim*

```
Proc Risk;
Environment open = "&RiskEnv";
Simulation Model_Sim  Method = Montecarlo
                      Data = History
                      Interval = weekday
                      Errmod = normal
                      Seed = 12345
                      Ndraw = 1000
                      Generator = pseudo
                      Horizon = 1 ;
Environment save;
Run;
```

10.3 Credit risk analyses

One of the possible risks that financial companies are exposed to, is **credit risk**. It is defined as the change in portfolio value due to defaulting on an obligation by a counter party. Various **credit risk analyses** are available in Risk Dimensions. They are used to obtain information about the amount and type of credit risk the company is exposed to.

Some examples of analyses are **Current exposure analysis**, **Potential exposure** analysis and **Credit Rating migration**.

Current exposure analysis is used to calculate the credit exposure of the portfolio that the company holds, using the base case market state.

Potential exposure analysis is used to calculate the credit exposure of the predicted future values of the portfolio. Monte Carlo simulation is used to predict the future market states that are used to create the future portfolio values.

Credit risk migration is used to create more complicated credit risk analyses. Each of the counterparties in the open positions that are held is given a credit rating. This might be, for example a high, medium or low risk of default rating. The strength of credit risk migration is that it makes provision for the fact that the probability of a counterparty defaulting, may change over time. If the probability changes a sufficient amount of percentage points, the credit rating category will also change. The use of credit risk migrations leads to a sophisticated prediction of credit risk at future dates.

The credit risk analyses in Risk Dimensions is not discussed in further detail as it falls outside the focus point of this document. They may, however, be used to great effect in a credit risk management system.

10.4 General risk analyses

Risk Dimensions offers other risk analyses methods, in addition to the market and credit risk analyses that were already discussed or mentioned in this chapter. One of the general methods, namely **Descriptive Statistics analysis** is discussed in detail whilst an overview is given of the other two namely **Cash Flow analysis** and **Portfolio optimization analysis**.

A **Portfolio statistics structure** is used to calculate general, descriptive statistical measures of the data values of **instrument variables** like *Coupon* or *Premium*. The general measures include the minimum, maximum, mean and standard deviation. The calculated value of a measure, for example, the minimum may furthermore be stored in an **output variable** that is included in some output data sets. The descriptive statistics may also be used in postprice or postvar method programs for further analysis.

The ***Statistics*** statement in *Proc Risk* is used to create a portfolio statistics structure in a risk environment. The general form of this statement is:

```
Statistics  name
        stat-1   variable-name-1 = output-variable-name-1,
        stat-2   variable-name-2 = output-variable-name-2,
          …
        stat-n   variable-name-n = output-variable-name-n ;
```

The name of the portfolio statistics structure is specified in the ***name*** option. The set of specifications for each descriptive statistic, is separated by a comma. The following options are used to create the specifications:

stat-i

The statistical measure that is used is specified in this option. The most frequently used measures are the minimum (*min*), maximum (*max*), number of values (*N*), standard deviation (*std*) and mean (*mean*).

variable-name-i

The name of the instrument variable that is used in the calculation of the i'th descriptive statistic, is specified in this option.

output-variable-name-i

The name of the output variable that receives the calculated value of the i'th statistical measure, is specified in this option.

Program Code 10.13 is used to create the portfolio statistics structure named *Premium_Stats* in the *Casestudy_Env* risk environment. The structure is used to calculate the minimum, maximum, mean and standard deviation of the values contained in the instrument variable *Premium*. The output variables *min_Premium*, *max_Premium*, *mean_Premium* and *std_Premium* are created to refer to the calculated statistical measures.

Program Code 10.13: Portfolios statistics structure *Premium_Stats*

```
Proc Risk;
Environment open = "&RiskEnv";
Statistics Premium_Stats
      min Premium = min_Premium,
      max Premium = max_Premium,
      mean Premium = mean_Premium,
      std Premium = std_Premium;
Environment save;
Run;
```

The *Premium_Stats* structure is executed in Section 10.6. The results that are obtained by execution are discussed in Section 10.7.

Another general risk analysis structure, named **Cash Flow analysis** is used to manage the risk associated with the possible gaps between the rate of return earned on assets and the cost of liabilities that may arise. The difference between the net inflow of cash from assets and the net outflow of cash from liabilities for a given time period, is defined as **gaps**. A positive gap is a positive net cash flow and a negative gap a negative net cash flow, for a given time period. Liquidity gaps, interest rate gaps, as well as duration gaps for a certain time period, are some of the measurements that are calculated. A liquidity gap refers to the difference between the amount of assets and liabilities held. Interest rate gaps arise when the assets and liabilities that are held are indexed to different market related interest rates. If the market movements of the interest rates differ in size or direction, the interest rate gaps arise. Duration gaps arise when the sensitivities of the assets and liabilities to parallel changes in the yield curve are not matched. Cash flow analysis is a very useful analysis structure, but is not discussed further in detail.

The last general risk analysis structure that is discussed briefly is a **Portfolio optimization structure.** The structure is used to optimize the value of a statistical measure without violating certain user-defined constraints. The constraints may include other statistical measures or certain user-defined bounds within and across cross-classifications. A well known example of this structure is **mean-variance optimization**.

An example of an optimization problem for the case study company may be:

- Maximise the expected return on the portfolio held, by keeping the variance of return below a user-defined level and by not investing more than 40% of the total portfolio in equities.

Risk Dimensions support three methods of optimization, namely:

- Return versus standard deviation of return,
- return versus expected shortfall and
- return versus Value at Risk.

The expected shortfall of a portfolio value is defined as the expected portfolio value given that the loss is greater or equal to Value at Risk. The portfolio optimization structure is not discussed in further detail as it falls outside the focus of the document.

10.5 Cross-classifications

The execution of risk analyses creates results that supply information about how the value of the portfolio will change for certain changes in the values of the risk factor variables. The whole portfolio file is analyzed by default. Thus, the effect of the changes in the risk factor variable values on the value of the **whole portfolio**, is measured.

The next Risk Dimensions structure, called a **cross-classification** is used to create certain **sub-portfolios** of the portfolio file. The data values of instrument variables and certain system-defined variables are used to determine the sub-portfolios. The effect of the changes in the values of the risk factor variables will now also be measured on the values of the sub-portfolios, in addition to the value of the whole portfolio.

Consider the case study *Activegrowth* that has open positions in five different financial instruments or instrument types, namely equities, futures, interest rate swaps, options and government bonds. The company may use a cross-classification to create sub-portfolios that consist of only one instrument type each. Examples of sub-portfolios are, an equity portfolio, an options portfolio, a futures portfolio etc. Risk analyses will be executed on the whole portfolio, as well as the five sub-portfolios.

The ***CrossClass*** statement in *Proc Risk* is used to create a cross-classification structure in a risk environment. The names of the instrument variables and system-defined variables that are used to create the sub-portfolios are also specified in this statement. The general form of the statement is:

```
Crossclass name (variable list);
```

The name of the structure is specified in the ***name*** option.

One or more variable names may be listed in the ***variable list*** option. If more than one variable name is listed, then two methods exist whereby the sub-portfolio is created. These methods are subsequently discussed.

Suppose two variables, *variable1* and *variable2* are specified in the variable list. It is optional to separate the variables by an asterisk “*”. This implies that:

1. All combinations of *variable1* and *variable2* will be analysed separately,
2. all levels of *variable1*, summed over *variable2*, as well as all levels of *variable2* summed over *variable1*, will be analysed and
3. the complete portfolio will be analysed separately.

If the asterisk is omitted, the second option changes to only:

2. All levels of *variable1* summed over *variable2* will be analysed.

The other two options remain the same, regardless of the asterisk. More than one variable list can also be specified to create different reporting structures for a single *Crossclass* statement.

Program Code 10.14 is used to create the cross-classification structure, named *By_Insttype* in the *Casestudy_Env* risk environment. The data values of the system-defined variable, named *InstType* is used to create the sub-portfolios. Each one of the sub-portfolios consists of positions that are held in the same instrument type.

Program Code 10.14: The cross-classification named *By_Insttype*

```
Proc Risk;
Environment open = "&RiskEnv";
Crossclass By_Insttype (InstType);
Environment save;
Run;
```

Cross-classifications are used in Risk Dimensions projects that are discussed in the next section.

10.6 Projects

Various Risk Dimensions structures have been created from Chapter 3 to Section 10.5. The information in these structures is combined into a single Risk Dimensions structure called a **project**, in this section. The combination takes place during the creation of the project structure. The created project may then be activated. This leads to the execution of the calculations that are necessary in risk analyses. Calculations, such as the calculation of the portfolio value are done for the first time in the risk management system.

Projects collect information from the following Risk Dimensions structures:

- Portfolio files,
- market data sources, parameter matrices, transformation sets,
- risk analysis structures,
- risk factor models,
- cross-classifications,
- reports and
- other information.

The information is passed on to the Risk Dimensions engine that performs all the necessary calculations.

The ***Project*** statement in *Proc Risk* is used to create a project structure in a risk environment and has the following general form:

```
Project  name    Portfolio = portfolio-file    Options;
```

The name of the project structure is specified in the ***name*** option. The following general options are also available in this statement:

Analysis = analysis-list

A list of the names of the risk analysis structures that are used to perform risk analyses on the portfolio, is specified in this option.

Crossclass = name

The name of a cross-classification structure that is used to create sub-portfolio in the project, is specified. If this option is omitted, results are only calculated for the whole portfolio.

Currency = *currency-code*

The three letter currency code, for example *ZAR*, of the reporting currency, is specified in this option.

Data = market-data-list

The list of market data source names that provide the market data is specified. At least one market data source of type *Current* has to be specified.

Label = "label"

A descriptive label may be specified for the project structure.

Models = models-list

The list of fitted risk factor models that are used in Monte Carlo simulation, is specified.

Options = (options-list)

Analysis and output options may be specified in this option. An example of an analysis option is the specification of a value for *alpha*. This value is used as the quantile value in Value at Risk calculations. The names of the output data sets that contain information about the calculated risk analyses, are also specified in this option. The contents of the data sets are then also available in the *Analysis* tree of the GUI. Examples of the available data sets are discussed in Section 10.7.

Outpath = *"Path"*

The name and path of the folder where the analysis results folder is created in, is specified.

Outlib= SAS-library

The name of the SAS library corresponding to the output folder, is specified in this option.

Portfolio = portfolio-file

The name of the portfolio file that is used in the project, is specified in this required option.

Reports = report-list

A list of report structures is specified in this option. Reports are discussed in detail in Chapter 11.

Rftrans = transformation-set-list

The list of transformation sets that is necessary in the project, is specified.

Rundate = date

The base date or date of valuation of the project is specified. The date for example, 1 January 2004 is written as '1jan2004'd.

Statistics = name

The name of the portfolio statistics structure that calculates descriptive statistics for the instrument variables, is included in this option.

Other options are also available in the *Project* statement, but are not discussed as they are not necessary in the context of this document.

The ***Runproject*** statement is used to execute the analyses that are specified in Risk Dimensions projects. The general form of this statement is:

```
Runproject    name-of-project    Options;
```

The *Data*, *Date*, *Currency*, *Options*, *Out*, *Outpath*, *Outlib*, *Portfolio* and *Report* options may be specified in this statement. These options are exactly the same as the corresponding options in the *Project* statement that were discussed above. If values are specified in these options, they replace the values of the corresponding options in the *Project* statement.

Other options available in the *Runproject* statement enable user to control the processing of the project. The project may be paused at an intermediate step and the results that have been created thus far, may be stored as output. The processing may also be split between multiple *Runproject* statements and recombined in subsequent processing. This is, however, not done for the case study.

Program Code 10.14 is used to create a project structure, named *Casestudy_Proj* in the *Casestudy_Env* risk environment. Risk analysis structures, a cross-classification structure, a reporting currency, market data sources, risk factor models, various output options, a portfolio file, reports, transformation sets and a base date are also specified in the *Project* statement. A *Runproject* statement is used to execute the calculation of the various risk analyses in the *Casestudy_Env* risk environment.

Program Code 10.14: The project structure named *Casestudy_Proj*

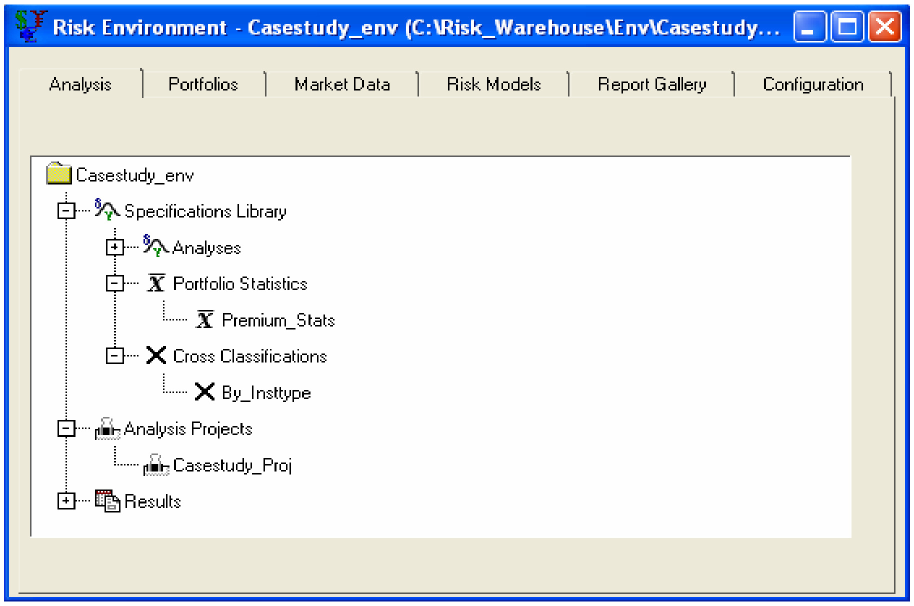
```
Proc Risk;
Environment open = "&RiskEnv";
Project Casestudy_Proj
            Analysis= (Sensit Equity_Curve Rate_Curve Scenario1 Stress1
                       Equity_Rate_surface Delta_Sim Hist_Sim Cov_Sim
                       Model_Sim Scen_Sim)
            Crossclass = By_Insttype
            Currency = ZAR
            Data = (Current History Covar )
            Models = (PC_Prin1 PC_Prin2 PC_Prin3 Ret_ASA Ret_AGL Ret_SLM
                      Ret_SOL Ret_ISC Ret_OML)
            Options = (alpha= 0.05 instvals simstates allstates )
            Out = Output
            Outlib = Output
            Outpath = "&Riskpath"
```

Program Code 10.14 Continues…

```
            Portfolio = All_deals_File
            Report = Summary_Report
            Rftrans = (Eigenvectortrans Modzerotrans)
            Rundate='13May2004'd
            Statistics = Premium_Stats;
Runproject Casestudy_Proj ;
Environment save;
Run;
```

The Risk Dimensions structures that were created in this chapter are viewed in the ***Analysis* tree** of the **GUI**. The risk analysis structures, portfolio statistics structures and cross-classifications are viewed under the *Specification Library* option in the *Analysis* tree. The risk analyses are viewed under the *Analyses* option, the statistics structure under the *Portfolio Statistics* options and the cross-classifications under the *Cross Classifications* option. The project structures that are created are viewed under the *Analysis Projects* option in the *Analysis* tree.

Figure 10.1 is used to view the Risk Dimensions structures that were created in Sections 10.2 to 10.6 in the *Casestudy_Env* risk environment.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy...
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration

```
Casestudy_env
  Specifications Library
    Analyses
    Portfolio Statistics
      Premium_Stats
    Cross Classifications
      By_Insttype
  Analysis Projects
    Casestudy_Proj
  Results
```

Figure 10.1: The Analysis tree of the *Casestudy_Env* risk environment

10.7 Risk analysis results and output data sets

The execution of a project structure, for example *Casestudy_Proj,* leads to the creation of **risk analysis results**.

Three types of risk analysis results are created, namely:

- SAS data sets, called **output data sets**,
- **graphical illustrations** and
- an additional feature, named **relative information measures**.

These three types of results are discussed in Sections 10.7.1, 10.7.2 and 10.7.3. The results of the execution of *Casestudy_Proj* are discussed in Section 10.7.4.

10.7.1 Output data sets

Various SAS data sets, called **output data sets** are created by the execution of a project structure in Risk Dimensions**.** The output data sets are grouped in a SAS library, for example *Output*. The output data sets may be viewed by the following two methods:

- The output data sets are listed under the *Data Files* option of the *Results* option in the ***Analysis* tree** of the **GUI**. A data set may be viewed by selecting the appropriate name in the *Data Files* option.

- The output data sets may also be viewed by selecting the name of an output data set in the **Explorer window** of the SAS window environment. The output data sets are grouped in a SAS library, for example *Output*, in this window.

Some of the most frequently used output data sets are:

Allprice

The value of each instrument type for each market state that is generated by the project execution, is contained in this data set.

Errors

The instrument types that could not be properly priced are contained in this data set. If there are no errors, this data set is not created.

Instvals

This data set contains one observation for each instrument type in the portfolio, that is valued for the base case market state. The sensitivities of the instrument values relative to the risk factor variables are also stored in the data set if a sensitivity analysis is executed.

Mktrates

The base case market state forms the one and only observation of this data set.

Mktstate

The data set contains one observation for each market state that is used to value the portfolio. The market states are created by the risk analyses that are included in the project.

Shocks

The simulated random shocks that are generated by the simulation analyses are stored in this data set.

Simdens

This data set contains the probability density estimate results of all the specified simulation analyses.

Simrfs

Summary statistics of the simulated risk factor variable values are contained in this data set.

Simrfim

Simulation risk factor information measures are included in this data set.

Simstat

This data set contains statistical measures of the simulated probability distributions of all the simulation analyses.

Simstate

The simulated market states that are generated by the simulation analyses forms this data set.

Simvalue

This data set contains the portfolio value for each replication of the simulations.

Summary

The base case mark-to-market value, the instrument variable statistics results (if requested) and the sensitivities of the portfolio value (if requested) are contained in this data set. One observation is made for each cross-classification group in this data set.

The *Summary*, *Simdens* and *Simstat* output data sets are created by default during the execution of a project. The other data sets are created by specifying the names of the data sets in the ***Options*** option in the *Project* or *Runproject* statements. Note that in order to create the *Mktstate* and *Simstate* output data sets the *Allstates* and *Simstates* options have to be used respectively.

The output data sets that are used in the case study are discussed in Section 10.7.4.

10.7.2 Graphical illustrations

The graphical illustrations that may be created by the execution of a project include:

- Profit/Loss curves,
- Profit/Loss surfaces and
- the graphical illustrations of the distribution function of the portfolio value under Scenario, Historical, covariance-based Monte Carlo and model-based Monte Carlo simulation.

The available graphical illustrations are listed in the *Reports* option in the *Results* option in the *Analysis* tree of the GUI. A graphical illustration is viewed by clicking on the appropriate option in the *Reports* option. The graphical illustrations that are created in the case study are discussed in Section 10.7.4.

10.7.3 Risk factor information measures

Risk factor information measures are used to measure the relative importance of each one of the risk factor variables to the portfolio value. The measures are calculated for simulation analyses.

The measure is computed from the joint bivariate distribution of the risk factor variables and the portfolio value. A discrete bivariate distribution is used to estimate the continuous joint distribution. The distribution is estimated by classifying the simulated portfolio value and corresponding risk factor values into a 5-by-5 table. Each simulated risk factor value and corresponding portfolio value is classified as either *very low*, *low*, *moderate*, *high*, or *very high*. The classifications of the portfolio value are based on the 20'th, 40'th,60'th and 80'th percentiles of the unconditional distribution. The classifications of the risk factor variables are based on cut points that are calculated from the sample moments. The number of classifications in each cell of the 5-by-5 table is calculated to form a contingency table.

The formula for the risk information measure for this discrete joint distribution is defined as:

$$\sum_{i=1}^{5}\sum_{j=1}^{5} f(i,j)\log_2\left(\frac{f(i,j)}{f(i)f(j)}\right) \qquad (10.8)$$

where

$f(i,j)$ = the observed fraction of simulations in which the portfolio value falls in category i and the risk factor variable value falls in category j,

$f(i)$ = the fraction of simulations were the portfolio value falls in category i,

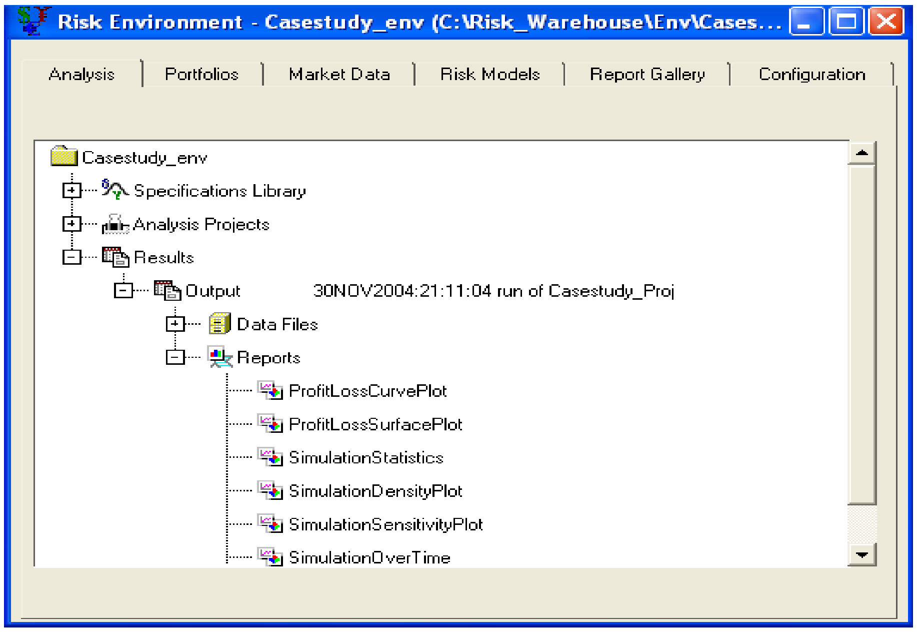
$f(j)$ = the fraction of simulations where the risk factor variable values falls in category j.

It further follows that $f(i)f(j)$ is the joint frequency for the (i,j) cell of the table that would be expected if the two variables were statistically independent. The relative information is therefore the expectation of the log ratio of the observed joint frequency to the joint frequency under independence. The use of this measure is illustrated for the simulation analyses of the case study, in Section 10.7.4.

10.7.4 Risk analysis results of the case study

The risk analysis results that are created in the *Casestudy_Env* risk environment are discussed in this section. The results are listed under the *Results* option in the *Analyses* tree of the GUI. This is illustrated in Figure 10.2. The results that are created by each risk analysis are discussed separately after this figure.

The name of the folder (*Output*) that is used to contain the output data sets is created under the *Results* option in the *Analysis* tree. Two options, namely the *Data Files* option and the *Reports* option are created automatically in this option. The output data sets are viewable under *Data Files* whilst the graphical illustrations and relative information measures are viewable under the *Reports* option. Some of the output data sets may be viewed under both options. Figure 10.2 is used to view the structure of the *Output* folder in the *Analysis* tree of the *Casestudy_Env* risk environment.



Casestudy_env
- Specifications Library
- Analysis Projects
- Results
  - Output 30NOV2004:21:11:04 run of Casestudy_Proj
    - Data Files
    - Reports
      - ProfitLossCurvePlot
      - ProfitLossSurfacePlot
      - SimulationStatistics
      - SimulationDensityPlot
      - SimulationSensitivityPlot
      - SimulationOverTime

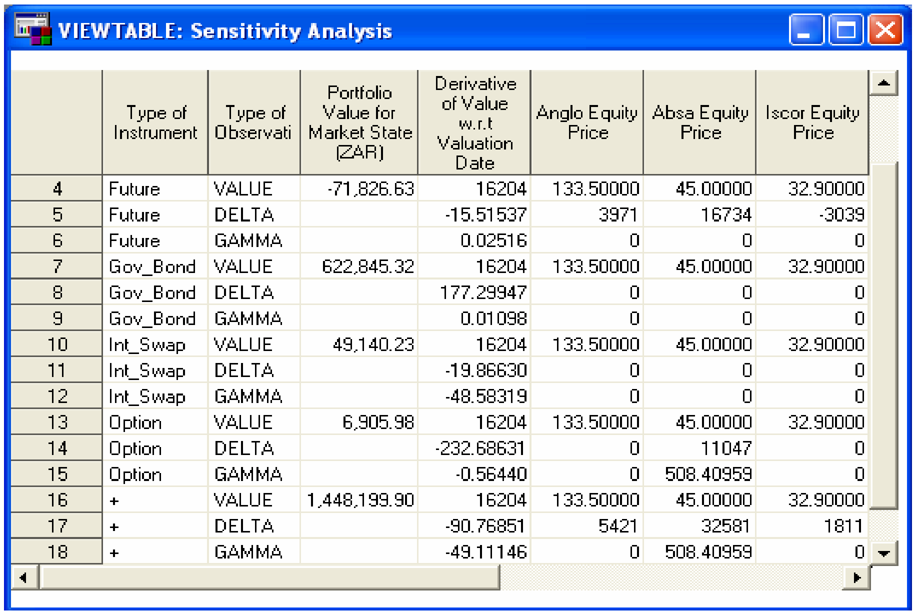
Figure 10.2: The *Results* option in the *Analysis* tree of *Casestudy_Env*

The output from the various risk analysis methods are discussed next.

Sensitivity analysis

The output data sets, named *Sens* and *Sens2* are created by the execution of a **sensitivity analysis structure** for example, the *Sensit* structure in the case study. The contents of these data sets may be viewed by selecting the *Sensitivity Analysis* and the *Sensitivity Analysis 2nd form* options under the *Data Files* option respectively. The explorer window may also be used to view the data sets.

Part of the information contained in the *Sens* data set is illustrated in Figure 10.3. For example, the change in the value of the whole portfolio for a one rand change in the Anglo equity price, is R 5,421. The change in the value of the sub-portfolio of futures, for a one rand change in the value of the Anglo equity price is R 3,971.

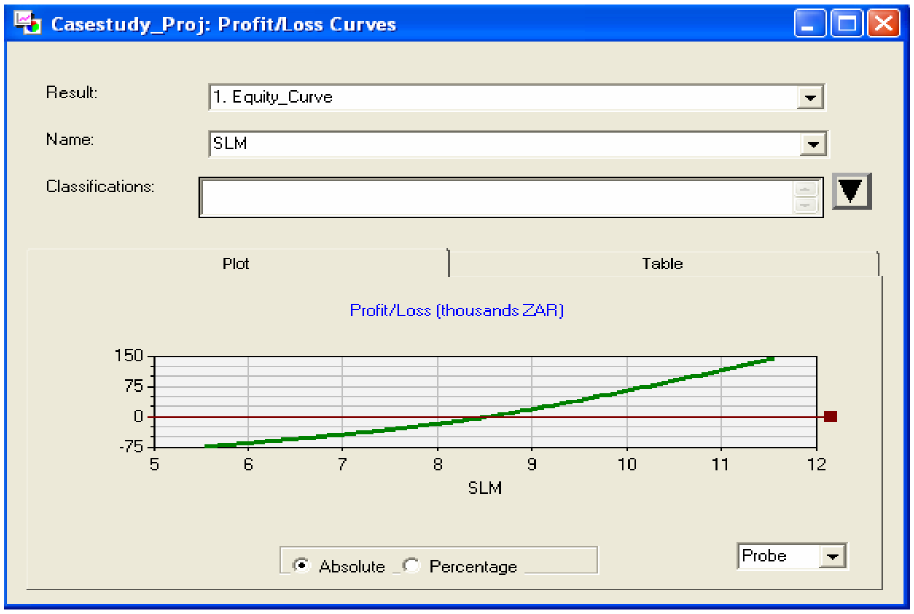


	Type of Instrument	Type of Observati	Portfolio Value for Market State (ZAR)	Derivative of Value w.r.t Valuation Date	Anglo Equity Price	Absa Equity Price	Iscor Equity Price
4	Future	VALUE	-71,826.63	16204	133.50000	45.00000	32.90000
5	Future	DELTA		-15.51537	3971	16734	-3039
6	Future	GAMMA		0.02516	0	0	0
7	Gov_Bond	VALUE	622,845.32	16204	133.50000	45.00000	32.90000
8	Gov_Bond	DELTA		177.29947	0	0	0
9	Gov_Bond	GAMMA		0.01098	0	0	0
10	Int_Swap	VALUE	49,140.23	16204	133.50000	45.00000	32.90000
11	Int_Swap	DELTA		-19.86630	0	0	0
12	Int_Swap	GAMMA		-48.58319	0	0	0
13	Option	VALUE	6,905.98	16204	133.50000	45.00000	32.90000
14	Option	DELTA		-232.68631	0	11047	0
15	Option	GAMMA		-0.56440	0	508.40959	0
16	+	VALUE	1,448,199.90	16204	133.50000	45.00000	32.90000
17	+	DELTA		-90.76851	5421	32581	1811
18	+	GAMMA		-49.11146	0	508.40959	0

Figure 10.3: The output data set *Sens*

Profit/Loss curves

The execution of a **Profit/Loss curves structure** creates an output data set, named *Plcurve,* as well as a set of graphical illustrations in the *Reports* option. The output data set *Plcurve* is stored in the *Output* library and is viewed either by electing the *Profit/Loss curves* option in the *Data Files* option or using the explorer window. The graphs of the profit/loss curves are viewed by selecting the *ProfitLossCurvePlot* option in the *Reports* option. One of the six profit/loss curves, namely the changes in the portfolio value for a grid of values of the *SLM* risk factor variable is viewed in Figure 10.4.


Casestudy_Proj: Profit/Loss Curves

Result: 1. Equity_Curve

Name: SLM

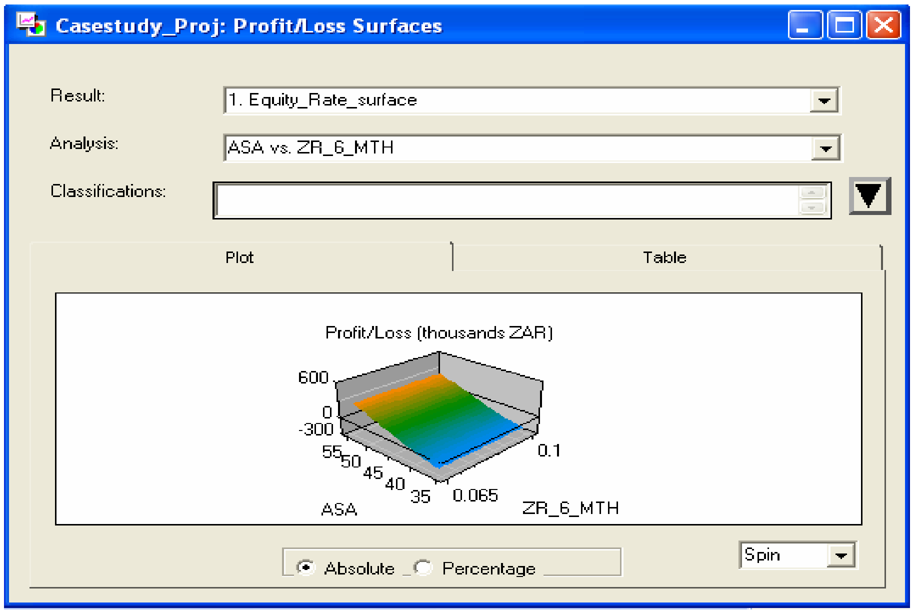
Classifications:

Figure 10.4: The *SLM* Profit/Loss curve in the *Equitycurve* set

The information about the grid points used in the plot is accessed by selecting the *Table* option in Figure 10.4. The change in portfolio value might also be specified as percentage and not the absolute value. As the company holds a net long position in *Sanlam* equities, the profit increases as the value of *SLM* increases.

Profit/Loss surfaces

The execution of a **Profit/Loss surface analysis** also creates an output data set and a graphical illustration. The output data set, named *Plsurf*, is stored in the *Output* library. It is viewed, either by selecting the *Profit/Loss Surfaces* option in the *Data Files* option or by using the Explorer window. The Profit/Loss surface plot is viewed by selecting the *ProfitLossSurfacePlot* option in the *Reports* option. The *Equity_Rate_surface* Profit/Loss surface is graphically illustrated in Figure 10.5.



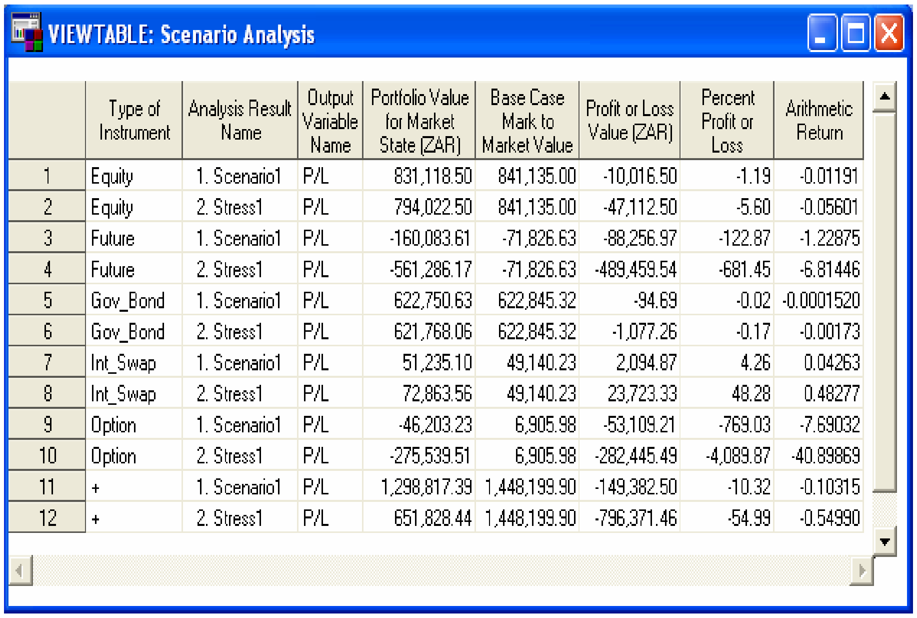
Result: 1. Equity_Rate_surface
Analysis: ASA vs. ZR_6_MTH
Classifications:
(● Absolute ○ Percentage)
Spin

Figure 10.5: The *Equity_Rate_surface* Profit/Loss surface

It is already known that an increase in either *ASA* or *ZR_6_MTH* leads to an increase in the profit of the portfolio. Figure 10.3 further illustrates that *ASA* has a bigger influence on the portfolio profit than *ZR_6_MTH*.

Scenario analysis and stress testing

An output data set, named *Scen* that is grouped in the *Output* library is created during the execution of **scenario analysis** and **stress testing**. The contents of the data set are viewed by clicking on the *Scenario Analysis* option under the *Data Files* option. The explorer window may also be used. The data set is viewed in Figure 10.6.



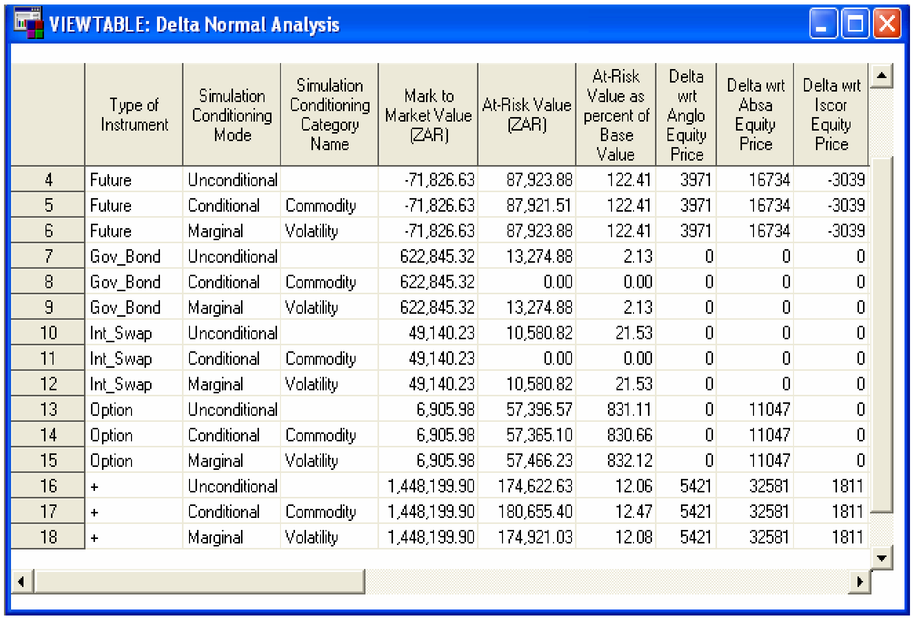
	Type of Instrument	Analysis Result Name	Output Variable Name	Portfolio Value for Market State (ZAR)	Base Case Mark to Market Value	Profit or Loss Value (ZAR)	Percent Profit or Loss	Arithmetic Return
1	Equity	1. Scenario1	P/L	831,118.50	841,135.00	-10,016.50	-1.19	-0.01191
2	Equity	2. Stress1	P/L	794,022.50	841,135.00	-47,112.50	-5.60	-0.05601
3	Future	1. Scenario1	P/L	-160,083.61	-71,826.63	-88,256.97	-122.87	-1.22875
4	Future	2. Stress1	P/L	-561,286.17	-71,826.63	-489,459.54	-681.45	-6.81446
5	Gov_Bond	1. Scenario1	P/L	622,750.63	622,845.32	-94.69	-0.02	-0.0001520
6	Gov_Bond	2. Stress1	P/L	621,768.06	622,845.32	-1,077.26	-0.17	-0.00173
7	Int_Swap	1. Scenario1	P/L	51,235.10	49,140.23	2,094.87	4.26	0.04263
8	Int_Swap	2. Stress1	P/L	72,863.56	49,140.23	23,723.33	48.28	0.48277
9	Option	1. Scenario1	P/L	-46,203.23	6,905.98	-53,109.21	-769.03	-7.69032
10	Option	2. Stress1	P/L	-275,539.51	6,905.98	-282,445.49	-4,089.87	-40.89869
11	+	1. Scenario1	P/L	1,298,817.39	1,448,199.90	-149,382.50	-10.32	-0.10315
12	+	2. Stress1	P/L	651,828.44	1,448,199.90	-796,371.46	-54.99	-0.54990

Figure 10.6: The output data set named *Scen*

The portfolio value under *Scenario1* and *Stress1* will be respectively R149,382.50 and R796,371.46 less than the current portfolio value.

Delta-Normal analysis

The execution of a **Delta-Normal analysis** structure creates an output data set named *Dvar* that are grouped in the *Output* library. The contents of the data set are viewed by either selecting the *Delta-Normal Analysis* option in the *Data Files* option or by using the explorer window. The contents of the data set are illustrated in Figure 10.7.



VIEWTABLE: Delta Normal Analysis

	Type of Instrument	Simulation Conditioning Mode	Simulation Conditioning Category Name	Mark to Market Value (ZAR)	At-Risk Value (ZAR)	At-Risk Value as percent of Base Value	Delta wrt Anglo Equity Price	Delta wrt Absa Equity Price	Delta wrt Iscor Equity Price
4	Future	Unconditional		-71,826.63	87,923.88	122.41	3971	16734	-3039
5	Future	Conditional	Commodity	-71,826.63	87,921.51	122.41	3971	16734	-3039
6	Future	Marginal	Volatility	-71,826.63	87,923.88	122.41	3971	16734	-3039
7	Gov_Bond	Unconditional		622,845.32	13,274.88	2.13	0	0	0
8	Gov_Bond	Conditional	Commodity	622,845.32	0.00	0.00	0	0	0
9	Gov_Bond	Marginal	Volatility	622,845.32	13,274.88	2.13	0	0	0
10	Int_Swap	Unconditional		49,140.23	10,580.82	21.53	0	0	0
11	Int_Swap	Conditional	Commodity	49,140.23	0.00	0.00	0	0	0
12	Int_Swap	Marginal	Volatility	49,140.23	10,580.82	21.53	0	0	0
13	Option	Unconditional		6,905.98	57,396.57	831.11	0	11047	0
14	Option	Conditional	Commodity	6,905.98	57,365.10	830.66	0	11047	0
15	Option	Marginal	Volatility	6,905.98	57,466.23	832.12	0	11047	0
16	+	Unconditional		1,448,199.90	174,622.63	12.06	5421	32581	1811
17	+	Conditional	Commodity	1,448,199.90	180,655.40	12.47	5421	32581	1811
18	+	Marginal	Volatility	1,448,199.90	174,921.03	12.08	5421	32581	1811

Figure 10.7: The output data set named *Dvar*

The unconditional Value at Risk is R 174,622.33, the Conditional Value at Risk is R 180,655.40 and the Marginal Value at Risk is R 174,921.03.

Simulation analyses

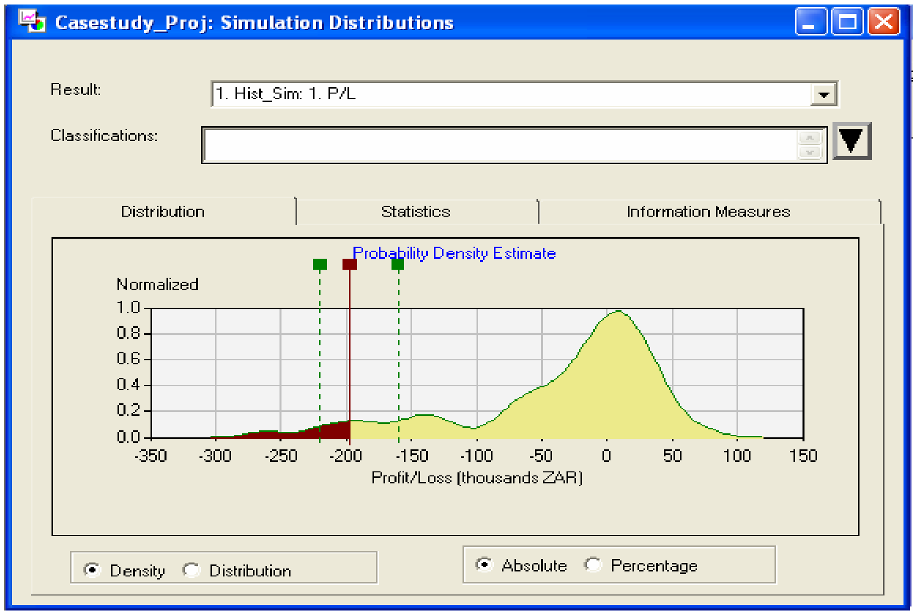
The four **simulation analyses** in the case study environment are historical simulation, covariance-based Monte Carlo simulation, model-based Monte Carlo simulation and scenario simulation. If these analyses are executed various output data sets, graphical illustrations and risk factor variable information measures are created. These three types of results are discussed separately for each one of the simulation analyses, later in this section.

The estimates of the probability density functions of all the four simulation analyses are contained in the *Simdens* output data set. The contents of this data set may be viewed by clicking on the *Simulation Distributions* option in the *Data Files* option. The output data set *Simstate* is activated by clicking on the *Simulation Statistics* option in the *Data Files* option. The simulation statistics of all four simulation analyses are available in this data set. The *Simrfim* data set may be activated by clicking on the *Simulation Risk Factor Information Measures* option in the *Data Files* option. The risk factor information measures of all the simulation analyses are contained in this data set. These output data sets may also be viewed using the explorer window.

The *SimulationDensityPlot* option in the *Reports* option is used to view the relevant contents of the *Simdens*, *Simrfim* and *Simstate* data sets separately for each of the simulation analyses. The contents of *Simdens* and *Simrfim* are graphically illustrated.

Historical simulation

Consider the execution of the **historical simulation** structure, *Hist_Sim*. The probability density function of the simulated portfolio value is viewed in Figure 10.8.



Result: 1. Hist_Sim: 1. P/L

Classifications:

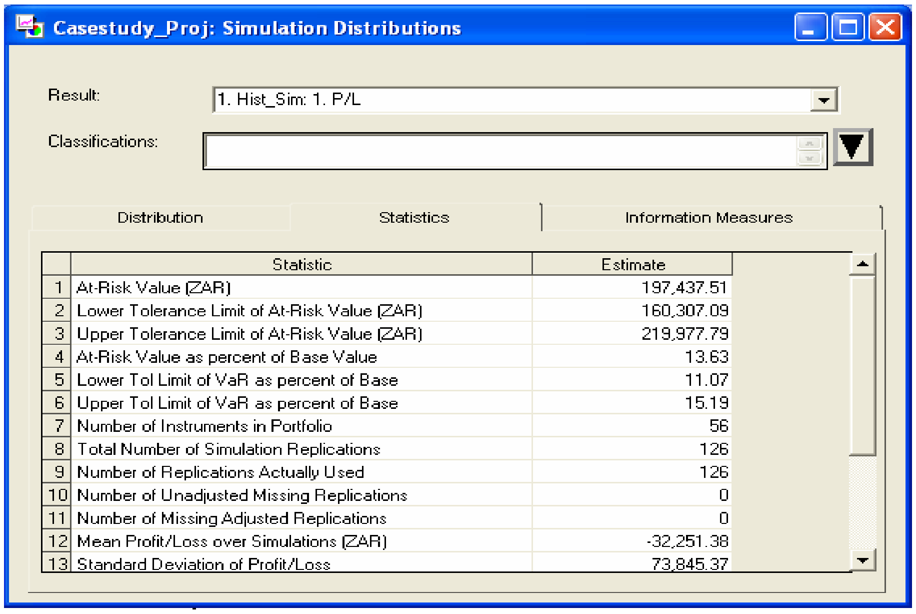
[✓ Density] Distribution

[✓ Absolute] Percentage

Figure 10.8: The probability density estimate of historical simulation

The fifth percentile of the distribution is shown with the red arrow. The Value at Risk estimate is equal to the distance between this point and 0.

Figure 10.9 is used to view the relevant part of the *Simstate* output data set for historical simulation. This is viewed by clicking on the *Statisitics* tab.

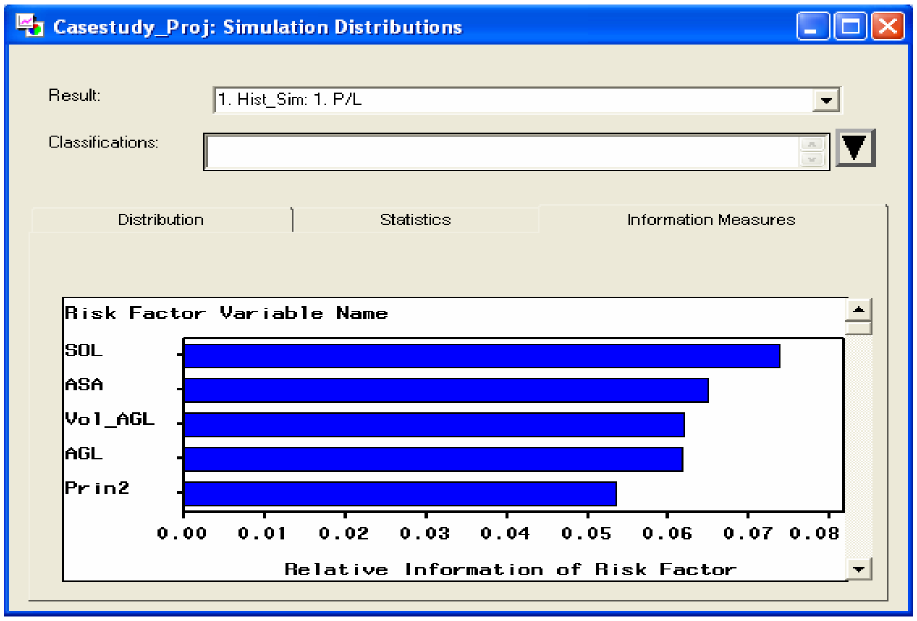


	Statistic	Estimate
1	At-Risk Value (ZAR)	197,437.51
2	Lower Tolerance Limit of At-Risk Value (ZAR)	160,307.09
3	Upper Tolerance Limit of At-Risk Value (ZAR)	219,977.79
4	At-Risk Value as percent of Base Value	13.63
5	Lower Tol Limit of VaR as percent of Base	11.07
6	Upper Tol Limit of VaR as percent of Base	15.19
7	Number of Instruments in Portfolio	56
8	Total Number of Simulation Replications	126
9	Number of Replications Actually Used	126
10	Number of Unadjusted Missing Replications	0
11	Number of Missing Adjusted Replications	0
12	Mean Profit/Loss over Simulations (ZAR)	-32,251.38
13	Standard Deviation of Profit/Loss	73,845.37

Figure 10.9: The simulation statistics of *Hist_Sim*

The 95% 1-day Value at Risk estimate, produced by historical simulation is R197,437.51. A confidence level is also calculated for the estimate. Various statistical measures are also available in the table illustrated in Figure 10.9.

The relative importance of the risk factor variables to the portfolio value in the historical simulation analyses, is viewed in Figure 10.10. The risk factor variables are listed in order of highest to lowest relative information measures. The variables *SOL*, *ASA*, *Vol_AGL*, *AGL* and *Prin2* have the largest influence on the portfolio value under historical simulation analysis. The relative information measures of the rest of the risk factor variables may be viewed by scrolling down.


Casestudy_Proj: Simulation Distributions
Distribution
Statistics
Information Measures
Risk Factor Variable Name
SOL
ASA
Vol_AGL
AGL
Prin2
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
Relative Information of Risk Factor

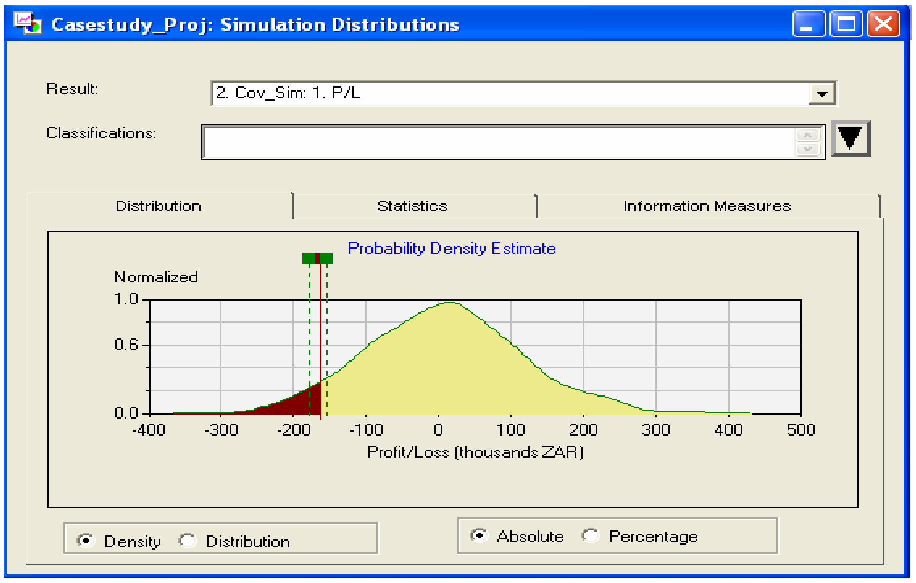
Result: 1. Hist_Sim: 1. P/L

Classifications:

Figure 10.10: The risk factor information measures of *Hist_Sim*

Covariance-based Monte Carlo simulation

Consider the execution of the **covariance-based Monte Carlo simulation** structure, named *Cov_Sim*. Figure 10.11 is used to view the estimate probability density function of the portfolio value.


Casestudy_Proj: Simulation Distributions
Distribution
Statistics
Information Measures
Probability Density Estimate
Normalized
1.0
0.6
0.0
-400
-300
-200
-100
0
100
200
300
400
500
Profit/Loss (thousands ZAR)

Result: 2. Cov_Sim: 1. P/L

Classifications:

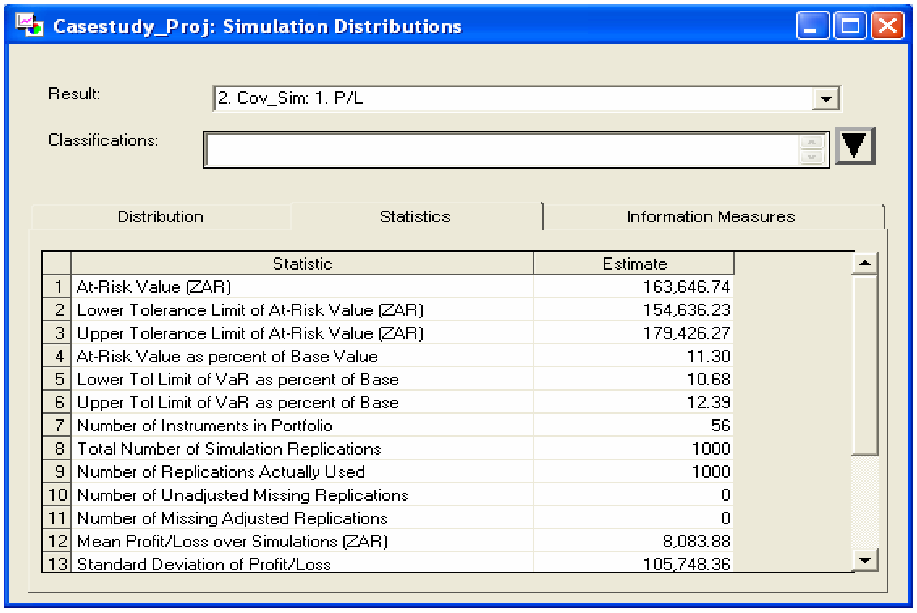
[✓ Density] Distribution

[✓ Absolute] Percentage

Figure 10.13: The probability density estimate of *Cov_Sim*

The fifth percentile of the distribution that is used in the estimation of VaR, is marked with a red arrow. The portfolio distribution may be standardized with respect to the base case portfolio value by clicking on the *Percentage* button. The estimated probability distribution function may be also be viewed by clicking on the *Distribution* button.

The part of the *Simstate* output data set that contains information about *Cov_Sim* is viewed in Figure 10.12. The Figure is viewed by clicking on the *Statisitics* tab.

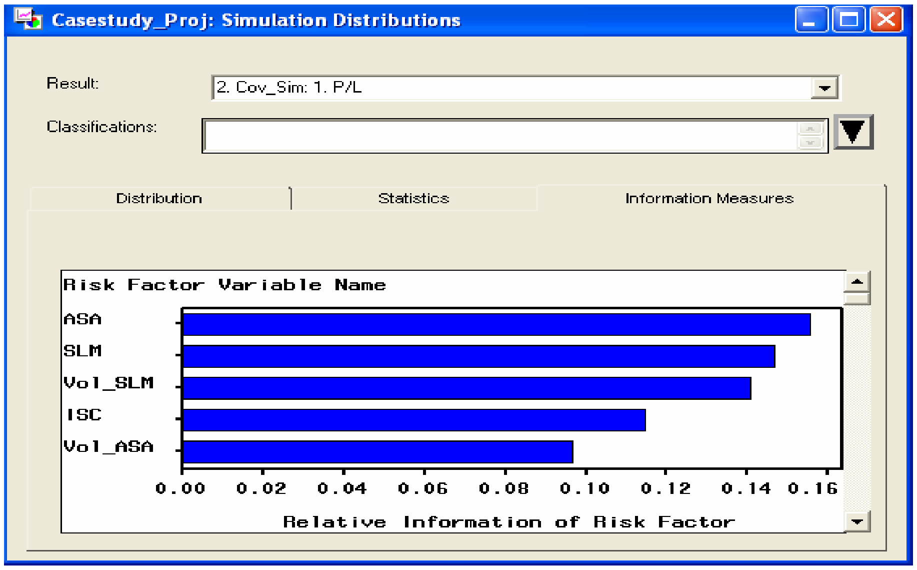


	Statistic	Estimate
1	At-Risk Value (ZAR)	163,646.74
2	Lower Tolerance Limit of At-Risk Value (ZAR)	154,636.23
3	Upper Tolerance Limit of At-Risk Value (ZAR)	179,426.27
4	At-Risk Value as percent of Base Value	11.30
5	Lower Tol Limit of VaR as percent of Base	10.68
6	Upper Tol Limit of VaR as percent of Base	12.39
7	Number of Instruments in Portfolio	56
8	Total Number of Simulation Replications	1000
9	Number of Replications Actually Used	1000
10	Number of Unadjusted Missing Replications	0
11	Number of Missing Adjusted Replications	0
12	Mean Profit/Loss over Simulations (ZAR)	8,083.88
13	Standard Deviation of Profit/Loss	105,748.36

Figure 10.12: The simulation statistics of *Cov_Sim*

The estimated 95% 1-day VaR is R163,646.74 which is 11.30% of the current portfolio value.

The risk factor information measures of *Cov_Sim* are illustrated in Figure 10.13. The risk factor variables that have the largest influence on the portfolio value under covariance-based Monte Carlo simulation are *ASA*, *SLM* and *Vol_SLM*.



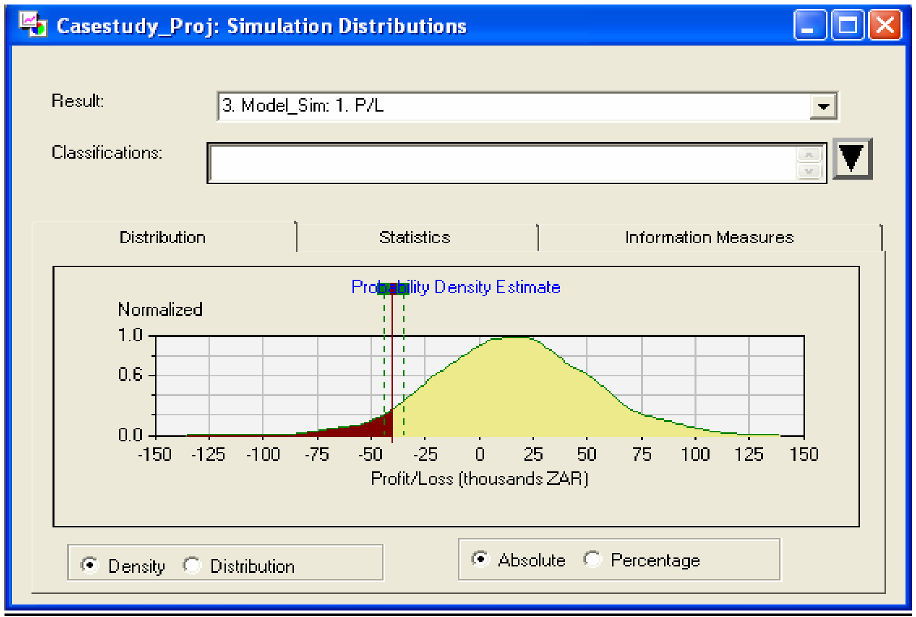
Result: 2. Cov_Sim: 1. P/L

Classifications:

Figure 10.15: The risk factor information measures of *Cov_Sim*

Model-based Monte Carlo simulation

Consider the execution of the model-based Monte Carlo simulation structure, named *Model_Sim*. The estimated probability density function of the portfolio value is viewed in Figure 10.14.



Result: 3. Model_Sim: 1. P/L

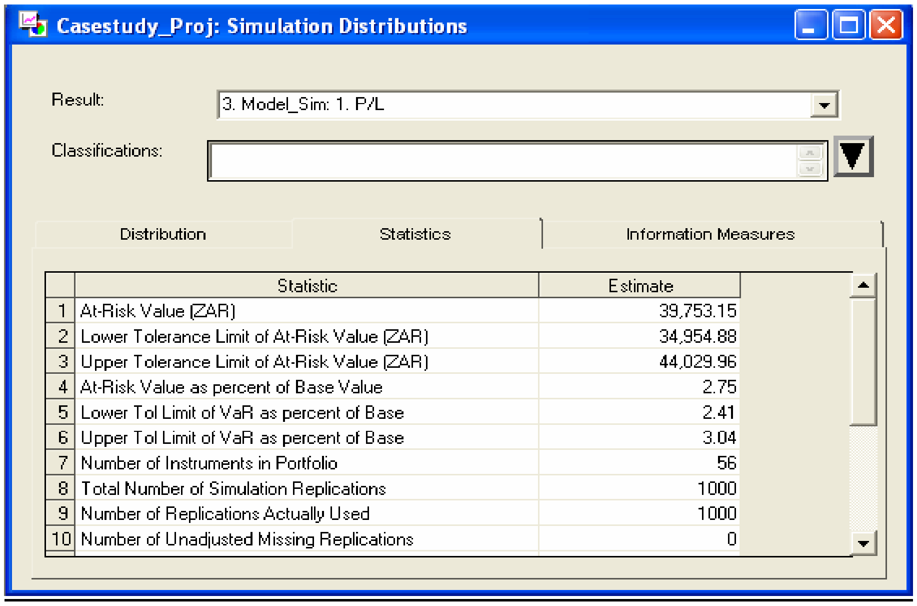
Classifications:

[✓ Density] Distribution

[✓ Absolute] Percentage

Figure 10.14: The probability density estimate of *Model_Sim*

The statistics of the *Model_Sim* simulation are illustrated in Figure 10.15.



Casestudy_Proj: Simulation Distributions

Result: 3. Model_Sim: 1. P/L

Classifications:

Distribution | Statistics | Information Measures

	Statistic	Estimate
1	At-Risk Value (ZAR)	39,753.15
2	Lower Tolerance Limit of At-Risk Value (ZAR)	34,954.88
3	Upper Tolerance Limit of At-Risk Value (ZAR)	44,029.96
4	At-Risk Value as percent of Base Value	2.75
5	Lower Tol Limit of VaR as percent of Base	2.41
6	Upper Tol Limit of VaR as percent of Base	3.04
7	Number of Instruments in Portfolio	56
8	Total Number of Simulation Replications	1000
9	Number of Replications Actually Used	1000
10	Number of Unadjusted Missing Replications	0

Figure 10.15: The simulation statistics of *Model_Sim*

The estimate of a 95% 1-day VaR is R 39,753.15 which is substantially smaller than the estimate of the other methods. The risk factor information measures of *Model_Sim* are illustrated in Figure 10.16.

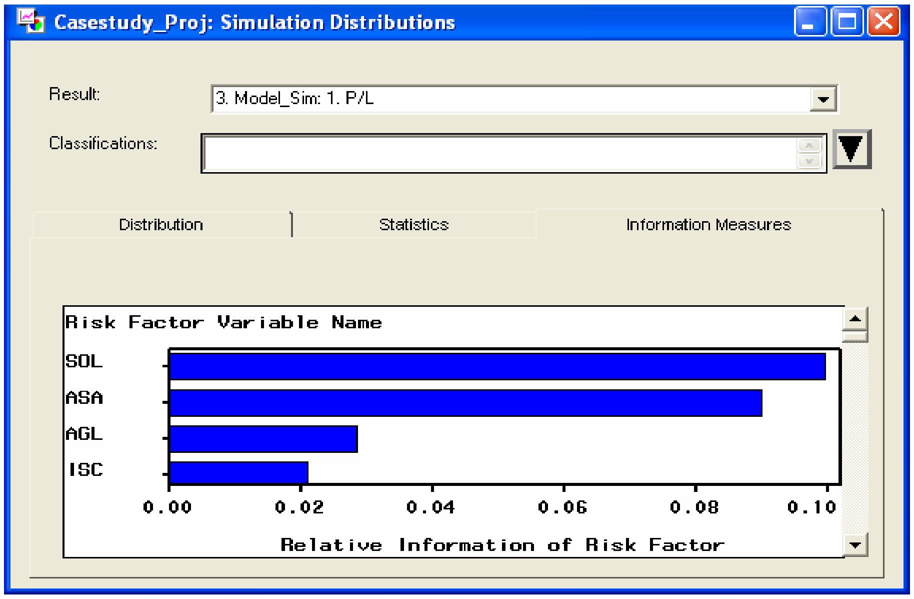
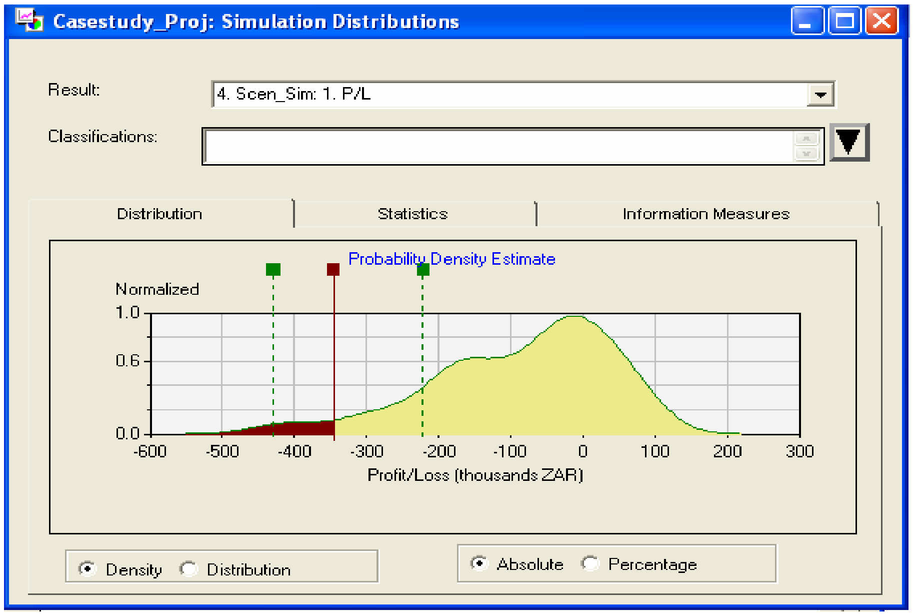


Figure 10.16: The relative information measures of *Model_Sim*

The risk factor variables that play a significant part in the portfolio value are *SOL* and *ASA*. The variables that have the third and fourth largest influence are also equities.

Scenario simulation

Consider the execution of the **scenario simulation** structure, named *Scen_Sim*. Figure 10.17 is used to view the estimated probability density function of the portfolio value.


Casestudy_Proj: Simulation Distributions
Distribution
Statistics
Information Measures
Probability Density Estimate
Normalized
1.0
0.6
0.0
-600
-500
-400
-300
-200
-100
0
100
200
300
Profit/Loss (thousands ZAR)

Result: 4. Scen_Sim: 1. P/L

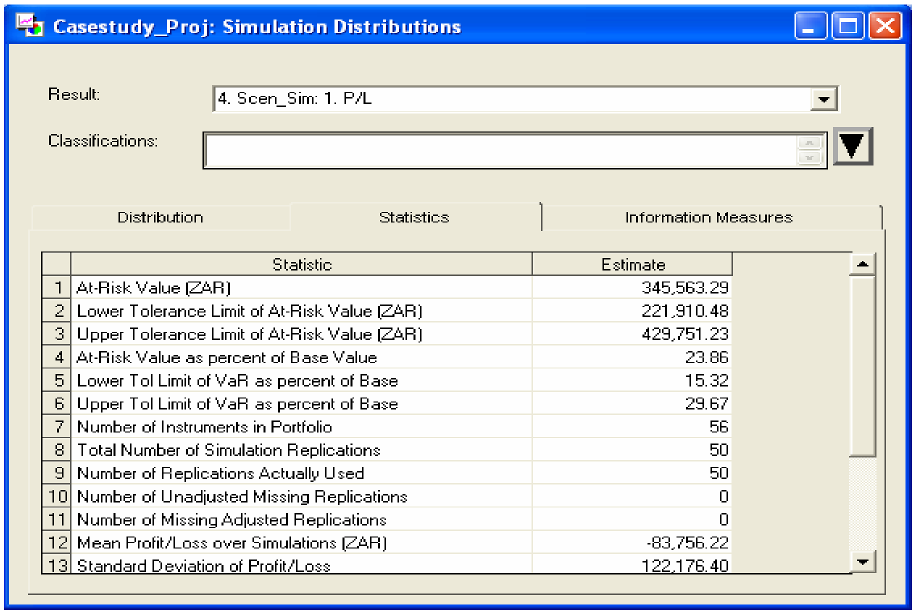
Classifications:

[✓ Density] Distribution

[✓ Absolute] Percentage

Figure 10.17: The probability density estimate of *Scen_Sim*

The simulation statistics of the *Scen_Sim* simulation analysis are viewed in Figure 10.18.



Casestudy_Proj: Simulation Distributions

Result: 4. Scen_Sim: 1. P/L

Classifications:

Distribution | Statistics | Information Measures

	Statistic	Estimate
1	At-Risk Value (ZAR)	345,563.29
2	Lower Tolerance Limit of At-Risk Value (ZAR)	221,910.48
3	Upper Tolerance Limit of At-Risk Value (ZAR)	429,751.23
4	At-Risk Value as percent of Base Value	23.86
5	Lower Tol Limit of VaR as percent of Base	15.32
6	Upper Tol Limit of VaR as percent of Base	29.67
7	Number of Instruments in Portfolio	56
8	Total Number of Simulation Replications	50
9	Number of Replications Actually Used	50
10	Number of Unadjusted Missing Replications	0
11	Number of Missing Adjusted Replications	0
12	Mean Profit/Loss over Simulations (ZAR)	-83,756.22
13	Standard Deviation of Profit/Loss	122,176.40

Figure 10.18: The simulation statistics of *Scen_Sim*

The 95% 1-day VaR of the portfolio value under the user-defined scenarios is R345,563.29. The risk factor information measures of *Scen_Sim* are viewed in Figure 10.19.

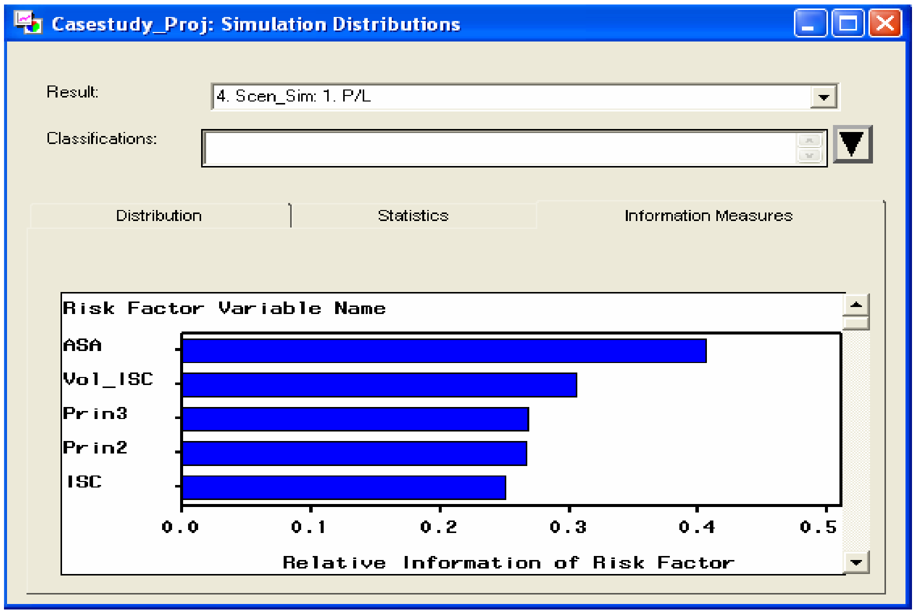


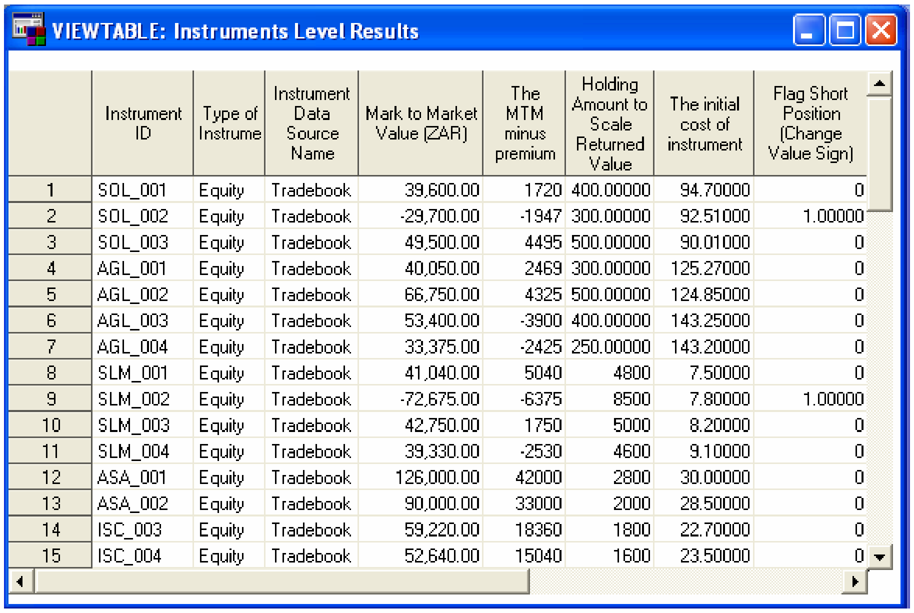
Figure 10.19: The relative information measures of *Scen_Sim*

The risk factor variable ASA has a very high risk factor information value. The other important variables are *Vol_ISC*, *Prin3*, *Prin2* and *ISC*.

Additional output data sets in *Casestudy_Env*

Consider the case study again. The following output data sets are also created in the *Output* library and are available in the *Data Files* option: *Instvals*, *Summary*, *Simstat*, *Simdens*, *Simrfim*, *Simstate* and *Mkstate*.

Figure 10.20 is used to view the contents of the ***Instvals*** data set. It is viewed by clicking on the *Instrument Level Results* option in the *Data Files* option in the *Analysis* tree or by using the explorer window.

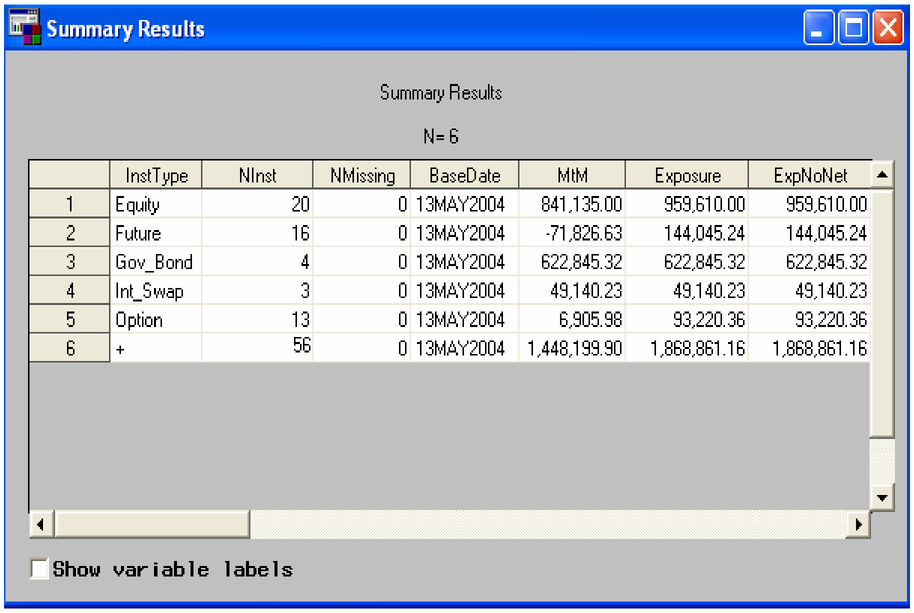


	Instrument ID	Type of Instrume	Instrument Data Source Name	Mark to Market Value (ZAR)	The MTM minus premium	Holding Amount to Scale Returned Value	The initial cost of instrument	Flag Short Position (Change Value Sign)
1	SOL_001	Equity	Tradebook	39,600.00	1720	400.00000	94.70000	0
2	SOL_002	Equity	Tradebook	-29,700.00	-1947	300.00000	92.51000	1.00000
3	SOL_003	Equity	Tradebook	49,500.00	4495	500.00000	90.01000	0
4	AGL_001	Equity	Tradebook	40,050.00	2469	300.00000	125.27000	0
5	AGL_002	Equity	Tradebook	66,750.00	4325	500.00000	124.85000	0
6	AGL_003	Equity	Tradebook	53,400.00	-3900	400.00000	143.25000	0
7	AGL_004	Equity	Tradebook	33,375.00	-2425	250.00000	143.20000	0
8	SLM_001	Equity	Tradebook	41,040.00	5040	4800	7.50000	0
9	SLM_002	Equity	Tradebook	-72,675.00	-6375	8500	7.80000	1.00000
10	SLM_003	Equity	Tradebook	42,750.00	1750	5000	8.20000	0
11	SLM_004	Equity	Tradebook	39,330.00	-2530	4600	9.10000	0
12	ASA_001	Equity	Tradebook	126,000.00	42000	2800	30.00000	0
13	ASA_002	Equity	Tradebook	90,000.00	33000	2000	28.50000	0
14	ISC_003	Equity	Tradebook	59,220.00	18360	1800	22.70000	0
15	ISC_004	Equity	Tradebook	52,640.00	15040	1600	23.50000	0

Figure 10.8: The output data set named *Instvals*

Information about each position that is held by the company, is recorded in a row.

Figure 10.21 is used to view the contents of the output data set, named ***Summary***. This is viewed by selecting the *Summary Results* option in the *Data Files* option, or by using the explorer window.



Summary Results

N= 6

	InstType	NInst	NMissing	BaseDate	MtM	Exposure	ExpNoNet
1	Equity	20	0	13MAY2004	841,135.00	959,610.00	959,610.00
2	Future	16	0	13MAY2004	-71,826.63	144,045.24	144,045.24
3	Gov_Bond	4	0	13MAY2004	622,845.32	622,845.32	622,845.32
4	Int_Swap	3	0	13MAY2004	49,140.23	49,140.23	49,140.23
5	Option	13	0	13MAY2004	6,905.98	93,220.36	93,220.36
6	+	56	0	13MAY2004	1,448,199.90	1,868,861.16	1,868,861.16

☐ Show variable labels

Figure 10.21: The *Summary* output data set

The information about each instrument type held, as well as the portfolio summary, is contained in this data set. The information includes mark-to-market values, as well as, the values of the output variables. Examples of the output variables are *Mean_Premium* that were created by the *Stats_Premium* portfolio statistics structure and *Daily_Profit* that was used in the pricing methods discussed in Chapter 7.

The user may also view the contents of the *Mkstate* output data set by clicking on the *All Analysis Market states* option in the *Data Files* option.

10.8 Two additional SAS statements

10.8.1 The *%Include* statement

The ***%Include*** statement may be used to include the program code of one SAS program , in another SAS program.

Suppose that the user has created a SAS program and has saved the program as a SAS file, for example *Program1.sas* in the *C:\Risk_Warehouse\Source* folder.

The following *%Include* statement in *Program2* includes the whole block of program code of *Program1*, in *Program2*.

```
%Include "C:\ Risk_Warehouse\Source\Program1.sas"
```

This statement may be very useful in the case study. The program code of each chapter may for instance be stored in a separate SAS program. The SAS programs may then be included in an additional program, that is executed.

10.8.2 The *Trace* statement

The ***Trace*** statement in *Proc Risk* may be used to trace the operations or calculations that are performed as a project is executed. The actual calculations that are performed are printed in the **output window**. This statement may be very useful to detect the problem if a pricing method does not give the correct answer. The *Trace* statement is created before the *Runproject* statement in *Proc Risk* and has the following general form:

```
Trace Methods = method-list Variable= Variable-list  Flow = Flow-list
```

Some of the most frequently used options that are available in this statement are subsequently discussed:

Methods = method-list

The actual calculations of the methods that are listed in this option are printed in the output window. The execution of a project, leads to the calculations. The values of *All* or *None* may also be specified.

Variable = variable-list

A list of risk factor variables may be specified in this option. The values of the risk factor variables are monitored as they change during the execution of a project.

Flow = Flow-list

Various options may be specified in this option and are not discussed in detail. The options are used to determine the calculations, of which method programs, are printed in the output window.

Level = Brief **|** *Detailed*

The detail of the trace of the calculations is specified in this option. If *brief* is specified, only assignments to variables that are within the method are displayed in the output window. If the *detailed* is specified, every mathematical calculation that is performed by the method is printed.

The *Trace* statement in Program Code 10.15 is used to trace the calculations of the *Option_Prc* pricing method, during the execution of the *Casestudy_Proj* project. The *Option_Prc* method is specified in the Method options, whilst the *level* option is set to *detailed*. The *Trace* statement is used before the *Runproject* statement in the same *Proc Risk* procedure.

Program Code 10.15: The *Trace* statement in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
Trace Methods = Option_Prc Level = detailed ;
Runproject Casestudy_Proj ;
Environment save;
Run;
```

The use of the *Trace* statement in Program Code 10.15 may be extended in the case study to include more method programs and risk factor variables.

10.9 Summary

Various **market risk analysis structures** were created in Section 10.2. Credit risk analyses and general risk analyses were briefly discussed in Sections 10.3 and 10.4. **Cross-classifications structures** were created in Section 10.5 to create sub-portfolios that are analysed separately by the risk analyses.

The creation and execution of **project structures** were discussed and illustrated in Section 10.6. The execution of a project leads to the creation of various **output data sets**. The contents of the data sets may be viewed and some may, in addition, be **graphical illustrated**. Although a lot of useful information is contained in the data sets, it is mixed with information that is not that relevant in risk management decisions.

The next step in the risk management system makes use of **reports** to obtain the useful information from the output data sets and to present it in an easily interpretable way. The information in the reports is then used in risk management decisions. Reports are discussed in the next chapter.

11

REPORTS

11.1 Introduction

The execution of risk analyses leads to the creation of various output data sets, graphical illustrations and risk factor information measures, as presented in Chapter 10. The information contained in these output data sets is essential for the risk management system, in fact, the producing of these results is the ultimate goal of the implementation of Risk Dimensions. The usefulness of this information, however, ranges from less to highly useful.

The information is also not presented very efficiently. Risk Dimensions **reports** provides the answer to this problem. Reports are used to extract user-defined information from the output data sets and to present the results in an easily interpretable way. Various different reports are available in Risk Dimensions, for example **batch reports**, **web-based reports** and **EIS reports**.

Batch reports are created from the output data sets and are presented in the output window of the SAS window environment. It is not able to generate any graphical illustrations.

Risk Dimensions also support **web-based reports**. This enables the sharing of the project results over the internet or intranet. Risk Dimensions do not have to be installed on a station, to view the report. Most of the internet browsers are able to view the report.

The third type of **report**, namely **EIS** is used together with multidimensional databases (MDDB's). Web-based reporting and EIS reports are very powerful techniques in processing project results, but are both advanced topics and are not discussed in this document.

Batch reports are created by the SAS procedure, named ***Proc Report*** from the output data sets. The reports are saved in separate SAS program files (.sas). The creation of these reports is discussed in detail in **Section 11.2**.

The ***Report*** **statement** in *Proc Risk* is used to **register** the batch reports that are created in Section 11.2, in risk environments. The registered reports are listed in the ***Reports*** **option** of *Project* statement (see Section 10.6). If the project structure is executed, the batch reports that are created may be viewed in the *Report Gallery* tree of the GUI. The registration and results of batch reports in the risk environment is discussed in Section 11.3. The case study is referred to, throughout the chapter. Some concluding remarks close the chapter.

In short, the three steps that are necessary to implement reports in a risk environment are:

1. The creation of batch reports by *Proc Report*, that is each saved in a separate SAS program (Section 11.2).
2. The registration of the batch reports in Risk Dimensions, using the *Report* statement in Proc Risk. (Section 11.3).
3. The inclusion of the registered reports in the *Reports* option of the *Project* statement in Section 10.6.

Batch reports are then generated in the risk environment during the execution of the project structure in Section 10.6.

The reports that are created in the case study are viewed in Section 11.4.

11.2 The SAS procedure *Proc Report*

The SAS procedure, named ***Proc Report*** is used to create batch reports from the data contained in SAS data sets. The reports are viewed in the *Output* window of the SAS window environment. It is a very flexible method and is used to present the data values in a customized way. It may, in addition to the existing data values, add subtotals and a grand total for the column values.

The general form of the *Proc Report* procedure is:

```
Proc Report    Data = SAS-data-set  Options;
Statements;
Run;
```

The name of the SAS data set (output data set) that contains the information that is used in the report, is specified in the ***Data*** option. Various options in the *Proc Report* statement and other statements in the procedure are available to customize the report. If no options and statements are used, all the observations and variables of the data set are printed in the report. The report willl also have the following characteristics:

- Each data value is displayed as it is stored in the data set,
- the variable names are used as the column headings in the report,
- a default width is used for the columns of the report,
- the character values are left-justified,
- the numeric values are right-justified and
- the observations of the report are in the same order as in the data set.

If the *nowd* option is specified in the *Proc Report* statement, the report is printed in the output window. If this option is omitted the report is printed in a new window in the SAS window environment. The *Headline* option may be used to underline the header of the columns whilst the *Headskip* option is used to add a blank line between the column headers and the first row of the report.

The variables or columns of the SAS data set that are used in the report are specified in the ***Column*** statement in *Proc Report*. The order of the listing of the variables determines the order in which they are displayed in the report.

The ***Define*** statement is used to customize the way that each variable is presented in the report. This statement is used to specify the following information for each column or variable in the report:

- The SAS format that is used to present the data values of the column in a certain way,
- the width of the column,
- the name of the column header,
- the alignment of the values in the column and
- the order of the observations in the column.

The general form of this statement is:

Define *variable-name / Usage Attributes-list*

A *Define* statement is used for each variable that is customized. The options that are available in the statement (*Usage* and Attributes) may be defined in any order.

Consider variables that are of character data type. The default value of *Usage* is *Display.* This entails that all the values of the variable are displayed in the report. The default value of the *Usage* option for numerical variables is *Analysis*. The *Usage* attribute may also be specified as *Group* for both character and numeric variables. This entails that all the observations that have the same data values of this variable are collapsed into a single observation in the report. More than one variable may be defined as *Group*, but all these variables must precede variables of other types. The *Usage* option may also be specified as *Order* for both character and numerical variables. The use of this option determines the order of the rows in the report. The default order is ascending. A descending order is obtained by including the word *Descending* in the *Attributes-list* of the *Define* statement. The *Order* option also suppresses the printing of repetitious values in the variable.

It follows that if the report contains:

- at least one display variable and no group variables, all the values of numerical variables are included in the report.
- only numerical values, the report displays only grand totals for the numeric variables.
- group variables, the sum of the numeric variables values are listed in the report for each group.

The calculation of the sum of the numerical values, may be replaced by using the following statistics in the *Usage* option of the *Define* statement:

• *N*	number of non-missing values,
• *Mean*	average of values in variable,
• *Max*	maximum value in variable and
• *Min*	minimum value in variable.

Various other attributes may be specified in the ***Attribute-list*** option of the *Define* statement. The ***Format*** option is used to specify the SAS format that is used to display the variable values. The width of the column in the report is specified in the ***Width*** attribute. One of three words, namely *Center*, *Left* and *Right* are used to specify the alignment of the variable values and header of the column. The header of the column is defined by typing a suitable name between quotation marks in the *Attribute-list* option. A split character may be used to split the column headers into more than one row in the report. The splitting character is defined by the *split* = *"character "*option in the *Proc Report* statement.

The report may be customized further by adding titles and footnotes. The ***Title*** statements and ***Footnote*** statements are used to create a series of title and footnotes respectively. These statements are global and may be used inside or outside of the *Proc* step.

The general form of the *Title* statement is:

```
Titlen   "Text";
```

The value of *n* may range from 1 to 10. The titles appear at the top of each page of the report. An unnumbered title refers to *Title1*. A title that is specified, is used for all subsequent output until the title is changed or cancelled. The title on line *n* and the titles on all the lines after it are cancelled by the following statement:

```
Titlen;
```

The *Footnote* statement is used to specify a line of text that is printed at the bottom of each page of the report. The general form of this statement is:

```
Footnoten  "Text"
```

The value of n may also range from 1 to 10 and an unnumbered footnote is equivalent to *Footnote1*. The footnotes that are specified are also used until they are cancelled or replaced by new footnotes. The following *Footnote* statement cancels the footnote statement for line *n* and all the lines with higher numbers:

```
Footnoten;
```

The ***Rbreak*** summary statement in *Proc Report* is used to create a default summary at the beginning or end of each report. The general form of this statement is:

```
Rbreak  location /  options;
```

The *location* option is specified as either *after* or *before*. If *after* is specified, the summary is made at the end of the report whilst if *before* is specified, it is placed at the beginning of the report. The line that is before or after the report is called the summary line. The *summarize* option is used to calculate summarizing statistics in the summary line for each numeric variable. The *DOL* and *DUL* option places a double line over and under each value that occurs in the summary line respectively. The *D* character may be omitted for each option, creating a single line.

The ***Break*** statement is used in a similar way as the *Rbreak* statement. It is used to produce a default summary at a change in the value of a group variable. This variable is known as the break variable in this context. The information contained in the summary is applicable to a set of observations that has the same value for the group variable. The *Break* statement has the following form:

```
Break   location   break-variable-name /  Options;
```

The name of the group variable that is used as break variable is specified in the *break-variable-name* option. The meaning of the *location* option and the additional options of the *Rbreak* statement are still applicable in this statement. In addition the *Skip* option may be used to write a blank line after the summary line.

The *Suppress* option is also used to omit the name of the break variable at each summary line.

Five batch reports are created to use in the *Casestudy_Env* risk environment. Program Code 11.1 is used to create a batch report that contains **mark-to-market information** about the whole portfolio and each sub-portfolio created by the *Insttype* cross-classification. The information is obtained from the *Instvals* output data set in the *Output* library. The *Column*, *Define*, *Rbreak* and *Title* statements are used. Various options are specified in these statements to customize the report.

Program Code 11.1: The Mark-to-market Report

```
Proc Report Data = Output.instvals
            nowd split = "*" headline headskip;
Column Insttype Value;
Define Insttype/"Type*of*instrument" width = 15 center group;
Define Value/"Mark to Market*Value (ZAR)" width = 15 center;
Rbreak after/summarize dol;
Title1 "Market Report: 13 May 2004";
Title2 "Portfolio Summary";
Run;
```

Program Code 11.1 is saved in a SAS file, named *Sumreport.sas* and is stored in the *Source* folder. This code generates the Portfolio summary table in Output 11.1, that is included later in this chapter.

Program Code 11.2 is used to create a report that will present the results of the execution of **sensitivity analysis**, named *Sensit* in a neat way. The data values in the report are obtained from the output data set *Sens2* that are grouped in the *Output* library. Various statements and options are used again to customize the report. Note that the sub setting *Where* statement (see Section 6.2.3) is used.

Program Code 11.2: The Sensitivity Analysis Report

```
Proc Report Data = Output.sens2
            nowd split = "*" headline headskip;
Column Name Sensitivity Gamma;
Define Name/"Risk factor*variable" width=15 center;
Define Sensitivity/ "Delta" width = 12 center format = comma10.2;
Define Gamma/ "Gamma" width = 12 center format = comma10.2;
Where Insttype = "+";
Title1 "Market Report: 13 May 2004";
Title2 "Sensitivity Analysis";
Run;
```

Program Code 11.2 is saved in a SAS file, named *Sensreport.sas* and is stored in the *Source* folder. This code generates the *Sensitivity Analysis* table in Output 11.1.

Program Code 11.3 is used to create a batch report to customize the information contained in the output data set *Scen*. The information about the execution of the **scenario analysis** structure *Scenario1* and the **stress testing** structure *Stress1* is contained in this data set. Various statements and options are used in *Proc Report* to customize the report.

Program Code 11.3: The Scenario analysis and stress testing report

```
Proc Report Data = Output.Scen
            nowd split="*" headline headskip;
Column Resultname Insttype Value Basecasevalue PL;
Define Resultname/"Analysis" width = 15 center order;
Define Insttype/"Instrument*Type" width = 12 center;
Define Value/"MtM*of*analysis" width = 12 center;
Define BaseCasevalue/"MtM" width = 12 center;
Define PL /"Profit*or*Loss" width = 11 center;
Break after Resultname / skip ol ul summarize suppress;
Title1 "Market Report: 13 May 2004";
Title2 "Scenario analysis and stress testing ";
Where Insttype ^= "+";
Run;
```

Program Code 11.3 is contained in the SAS file *Scenreport.sas* that is stored in the *Source* folder. This code is used to create the *Scenario analysis and stress testing* table in Output 11.1.

Program Code 11.4 is used to create a batch report about the output information from the **Delta-Normal analysis**, named *Delta_Sim*. The variable name *VaR* in the output data set *Dvar* in the *Output* library is renamed to *ValueatRisk* in the *Dvar* data set in the *Work* library. The values of this data set are used in the report. Various options and statements are used in *Proc Report* to customize the report.

Program Code 11.4: The Delta Normal Analysis Report

```
Data Work.Dvar (rename = (VaR = ValueatRisk));
Set output.Dvar;
Run;
Proc Report Data = Work.dvar
            nowd Split = "*" headline headskip;
Column Simulationmode Insttype MtM ValueatRisk VaRPct;
Define Simulationmode/"Simulation*mode" width = 13 center order;
Define Insttype/"Instrument*Type" width = 12 center;
Define MtM/"Mark-to-Market*(ZAR)" width = 14 center;
Define ValueatRisk/"Value at Risk" width = 12 center;
Define VaRpct / "VaR as*percentage*of MtM" width = 12 center;
Break after simulationmode / skip ol  suppress;
Title1 "Market Report: 13 May 2004";
Title2 "Delta Normal Analyses";
Run;
```

Program Code 11.4 is stored as a SAS file, named *Dnreport.sas* in the Source folder. This code generates the *Delta Normal Analyses* table in Output11.1

A report that contains the **Value at Risk** information of the portfolio is created in Program Code 11.5. The information contained in the output data set *Simstat* is used to create the report. Various statements and options in *Proc Report* are used to create the report.

Program Code 11.5: The Value at Risk Report

```
Data Output.Simstat (rename = (VaR = ValueatRisk));
Set output.Simstat;
Run;
Proc Report data = Output.Simstat
             nowd split="*" headline headskip;
Column Resultname Insttype MtM ValueatRisk VaRPct ES;
Define Resultname/"Simulation method" width=17 center order;
Define Insttype/"Instrument*Type" width =12 center;
Define MtM/"Mark*to*Market*(ZAR)" width =12 center;
Define ValueatRisk/"Value at Risk" width =12 center;
Define VaRpct / "VaR as*percentage*of MtM" width = 10 center;
Define ES /"Estimated*shortfall" width =10 center;
Break after Resultname / skip ol suppress;
Title1 "Market report: 13 May 2004";
Title2 "95% 1-day Value at Risk ";
Run;
```

Program Code 11.5 is saved in SAS file, named *Varreport* that is stored in the *Source* folder. This code generates the *95% 1-day Value at Risk* table in Output 11.1.

11.3 The registration of reports

The ***Report*** statement in *Proc Risk* is used to register a batch report in a risk environment. The names of the registered reports are included in the ***Reports* option** of ***Project* statement** (see Section 10.6). The execution of the project structure leads to the creation of the batch reports. The reports that are created are viewed in the *Report Gallery* of the GUI.

First consider the general form of the *Report* statement:

Report name   Type = “type” File = “Path\filename” Options;

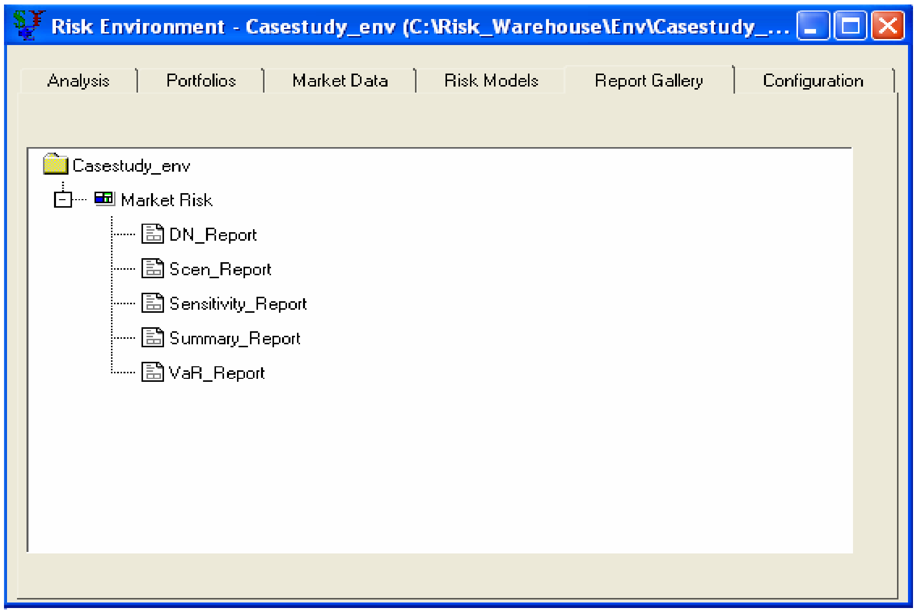
The name of the batch report is specified in the *name* option. The *type* option is used to group the reports in the GUI. It is usually specified as *“Market Risk”* or *“Credit Risk”*. The reports with similar types are grouped together in the GUI. The name and location of the SAS file that contains the batch report is specified in the *File* option. A descriptive label may also be specified for the report. Another option, namely the *Sampout* option is used to specify the folder that contains the output data sets that are used in the report.

Program Code 11.6 is used to register the batch reports that were created in Program Code 11.1 to 11.5 in the *Casestudy_Env* risk environment. For each report a *name*, the location of the batch report, a suitable report type and a folder that contains the output data sets are specified.

Program Code 11.6: Reports in *Casestudy_Env*

```
Proc Risk;
Environment open = "&RiskEnv";
Report Summary_Report
            File = "&Source\Sumreport.sas" /*Program Code 11.1*/
            Type = "Market Risk"
            Sampout = Output;
Report Sensitivity_Report
            File="&Source\Sensreport.sas" /*Program Code 11.2*/
            Type="Market Risk"
            Sampout=Output;
Report Scen_Report
            File="&Source\Scenreport.sas" /*Program Code 11.3*/
            Type="Market Risk"
            Sampout=Output;
Report DN_Report
            File="&Source\Dnreport.sas" /*Program Code 11.4*/
            Type="Market Risk"
            Sampout=Output;
Report VaR_Report
            File="&Source\Varreport.sas" /*Program Code 11.5*/
            Type="Market Risk"
            Sampout=Output;
Environment save;
Run;
```

Program Code 11.6 needs to be executed before the Program Code 10.14. The project structure *Casestudy_Proj* is created in this program code. The execution of Program Code 10.14 leads to the creation of batch reports in the *Report Gallery* tree of the GUI. Figure 11.1 is used to illustrate the contents of this tree for the *Casestudy_Env* risk environment. All the reports that were created had the *type* option specified as *Market Risk*. Thus, all the reports are grouped under the same option in the GUI.


Risk Environment - Casestudy_env (C:\Risk_Warehouse\Env\Casestudy_...
Analysis
Portfolios
Market Data
Risk Models
Report Gallery
Configuration

Casestudy_env
Market Risk
DN_Report
Scen_Report
Sensitivity_Report
Summary_Report
VaR_Report

Figure 11.1: The *Report Gallery* of *Casestudy_Env*

11.4 Reports in *Casestudy_Env*

Each batch report is viewed in the output window by clicking on the appropriate option in the *Report Gallery* tree of Figure 11.1. If all the reports are clicked, the information in Output 11.1 is obtained in the output window. The contents of this window may be printed.

Output 11.1: The Market Reports

```
                     Market Report: 13 May 2004
                        Portfolio Summary

                      Type
                       of          Mark to Market
                   instrument        Value (ZAR)
                 ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

                     Equity           841,135.00
                     Future           -71,826.63
                    Gov_Bond          622,845.32
                    Int_Swap           49,140.23
                     Option             6,905.98
                                 ===============
                                    1,448,199.90

                   Market Report: 13 May 2004
                      Sensitivity Analysis

             Risk factor
              variable          Delta          Gamma
            ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

                ASA          32,580.70         508.41
                AGL           5,420.87           0.00
                ISC           1,810.88           0.00
                SOL         -14,945.60          89.16
                SLM          36,942.98      11,422.97
                OML          19,246.49           0.00
              Vol_ASA        35,676.71     444,937.35
              Vol_SOL        -1,645.37     -53,994.04
              _date_            -90.77         -49.11

                  Market Report: 13 May 2004
              Scenario analysis and stress testing

                                   MtM                        Profit
                 Instrument         of                          or
   Analysis         Type         analysis         MtM          Loss
 ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

  1. Scenario1      Equity      831,118.50    841,135.00   -10,016.50
                    Future     -160,083.61    -71,826.63   -88,256.97
                   Gov_Bond     622,750.63    622,845.32       -94.69
                   Int_Swap      51,235.10     49,140.23     2,094.87
                    Option      -46,203.23      6,905.98   -53,109.21
                             ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒ
                             1,298,817.39  1,448,199.90  -149,382.50
                             ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒ
```

Output 11.1: continues …

```
   2. Stress1       Equity      794,022.50    841,135.00   -47,112.50
                    Future     -561,286.17    -71,826.63  -489,459.54
                   Gov_Bond     621,768.06    622,845.32    -1,077.26
                   Int_Swap      72,863.56     49,140.23    23,723.33
                    Option     -275,539.51      6,905.98  -282,445.49
                             ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒ
```

```
                                  651,828.44  1,448,199.90  -796,371.46
                                ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒ


                         Market Report: 13 May 2004
                            Delta Normal Analyses

                                                                VaR as
 Simulation     Instrument   Mark-to-Market    Value at     percentage
    mode           Type          (ZAR)           Risk         of MtM
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

 Conditional      Equity        841,135.00     93,920.48        11.17
                  Future        -71,826.63     87,921.51       122.41
                 Gov_Bond       622,845.32          0.00         0.00
                 Int_Swap        49,140.23          0.00         0.00
                  Option          6,905.98     57,365.10       830.66
                    +         1,448,199.90    180,655.40        12.47
                             ƒƒƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ

  Marginal        Equity        841,135.00     93,920.48        11.17
                  Future        -71,826.63     87,923.88       122.41
                 Gov_Bond       622,845.32     13,274.88         2.13
                 Int_Swap        49,140.23     10,580.82        21.53
                  Option          6,905.98     57,466.23       832.12
                    +         1,448,199.90    174,921.03        12.08
                             ƒƒƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ


Unconditional     Equity        841,135.00     93,920.48        11.17
                  Future        -71,826.63     87,923.88       122.41
                 Gov_Bond       622,845.32     13,274.88         2.13
                 Int_Swap        49,140.23     10,580.82        21.53
                  Option          6,905.98     57,396.57       831.11
                    +         1,448,199.90    174,622.63        12.06
                             ƒƒƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ
```

Output 11.1: continues …

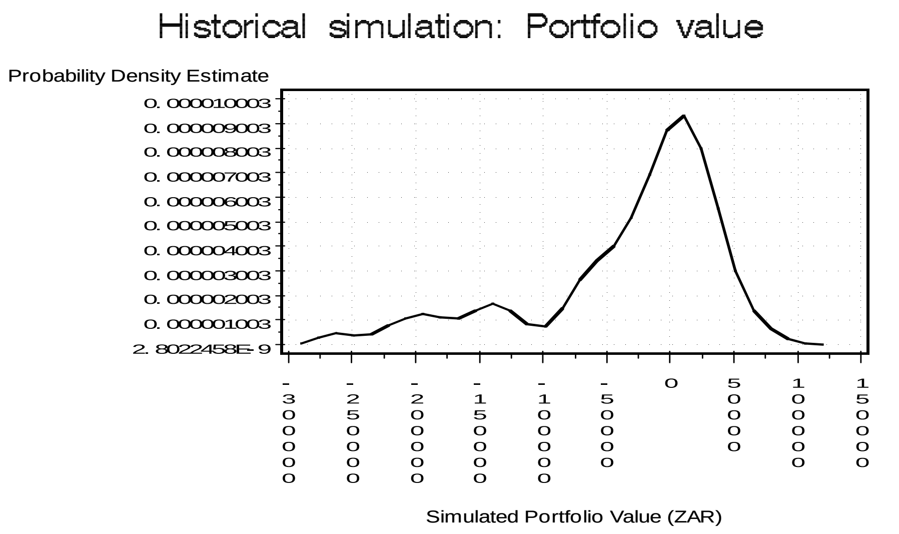
```
                               Market report: 13 May 2004
                                95% 1-day Value at Risk

                                        Mark
                                         to                   VaR as
                    Instrument         Market     Value at   percentage  Estimated
Simulation method      Type            (ZAR)        Risk       of MtM    shortfall
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

  1. Hist_Sim: 1.     Equity       841,135.00   187,900.00       22.34  220,137.85
                      Future       -71,826.63    37,992.13       52.89   47,001.85
                     Gov_Bond      622,845.32     2,439.06        0.39    2,884.32
                     Int_Swap       49,140.23     2,303.78        4.69    2,674.56
                      Option         6,905.98    21,099.33      305.52   26,366.75
```

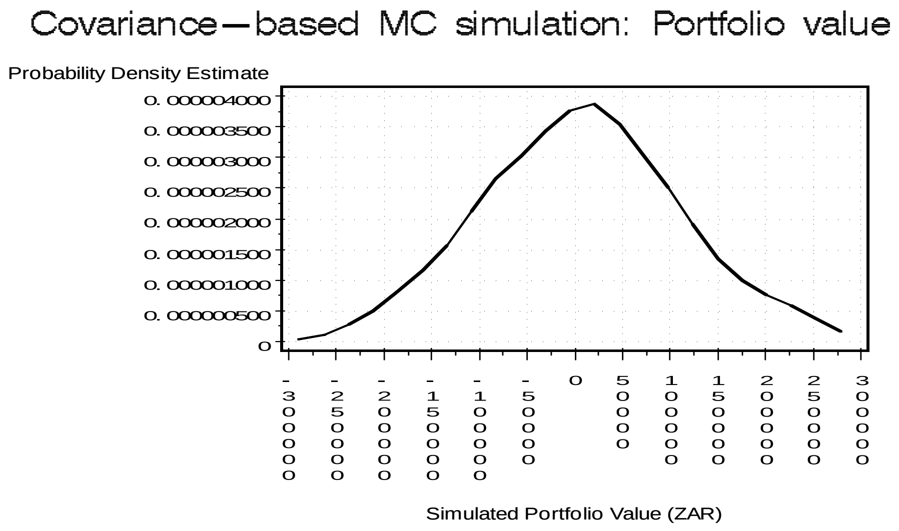
```
                         +        1,448,199.90    197,437.51       13.63  228,920.24
                                  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ

        2. Cov_Sim: 1.      Equity      841,135.00     92,554.44       11.00  116,546.93
                            Future      -71,826.63     85,419.13      118.92  104,859.00
                           Gov_Bond     622,845.32     12,722.85        2.04   15,960.04
                           Int_Swap      49,140.23     10,632.96       21.64   12,902.86
                            Option        6,905.98     49,742.82      720.29   62,974.89
                             +        1,448,199.90    163,646.74       11.30  201,310.00
                                  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ

        3. Model_Sim: 1     Equity      841,135.00     14,986.77        1.78   19,099.80
                            Future      -71,826.63     34,538.72       48.09   45,253.32
                           Gov_Bond     622,845.32      2,120.80        0.34    2,729.93
                           Int_Swap      49,140.23      1,994.84        4.06    2,480.13
                            Option        6,905.98      4,756.16       68.87    9,413.29
                             +        1,448,199.90     39,753.15        2.75   56,923.69
                                  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ

        4. Scen_Sim: 1.     Equity      841,135.00    218,838.50       26.02  225,680.00
                            Future      -71,826.63    181,679.13      252.94  214,945.34
                           Gov_Bond     622,845.32    -19,385.14       -3.11  -18,089.84
                           Int_Swap      49,140.23    -17,020.09      -34.64  -16,150.66
                            Option        6,905.98     89,994.73    1,303.14  109,092.48
                             +        1,448,199.90    345,563.29       23.86  406,045.44
                                  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ  ƒƒƒƒƒƒƒƒƒƒ
```

The information contained in Output 11.1 is easy to interpret and is used in risk management decisions. It is possible to create reports of this type at the end of every trading day.

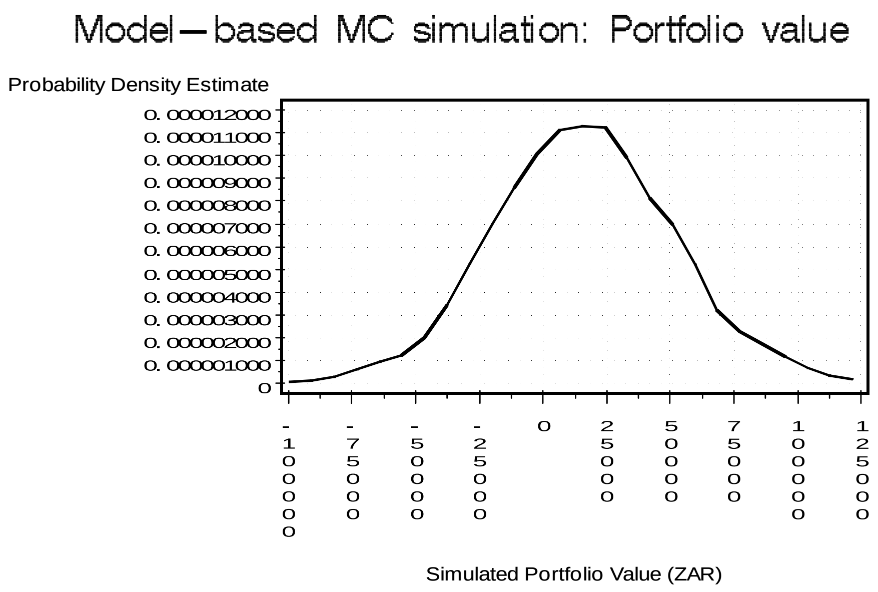
The output created by batch reports may be complemented by some **graphical illustrations** that are created in the **SAS Enterprise Guide** software program. The program was used to create an estimated probability density function of each of the simulation analyses in the case study risk environment. The information in the *Simdens* output data set was used. Enterprise Guide generates SAS program code that may be used in the enhanced editor window to create the graphical illustrations in the output window. The program code is not discussed, but it is important to take cognizance of this feature of the SAS package. The graphical illustrations follow in Output 11.2.

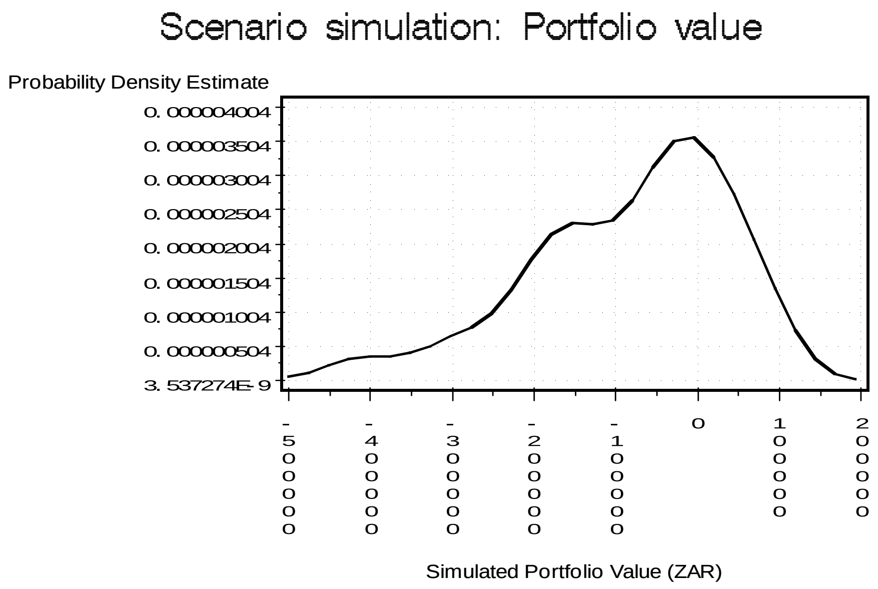
Output 11.2: The probability density estimates of the simulation analyses





Output 11.2: Continues …


Model-based MC simulation: Portfolio value
Probability Density Estimate
0.000012000
0.000011000
0.000010000
0.000009000
0.000008000
0.000007000
0.000006000
0.000005000
0.000004000
0.000003000
0.000002000
0.000001000
0
-100000
-75000
-50000
-25000
0
25000
50000
75000
100000
125000
Simulated Portfolio Value (ZAR)


Scenario simulation: Portfolio value
Probability Density Estimate
0.000004004
0.000003504
0.000003004
0.000002504
0.000002004
0.000001504
0.000001004
0.000000504
3.537274E-9
-500000
-400000
-300000
-200000
-100000
0
100000
200000
Simulated Portfolio Value (ZAR)

11.5 Summary

It is obvious from the content of this chapter that the creation of batch reports and graphical illustrations are the final step to present the final results of the risk management system in a user-friendly way. The producing of these final reports is, in fact, the ultimate goal of the implementation of Risk Dimensions. The reports contain all the necessary information that is used in risk management decisions and is easily updated at the end of each trading day.

12

CONCLUSION

SAS Risk Dimensions is a business tool that is used by financial institutions to create a risk management system. The system calculates various risk measures at the end of each trading day. The calculated values are presented in a report that is easily interpretable. The information contained in the report is used by senior management to make risk management decisions, during the next trading day.

It is relatively difficult and complex to create a suitable risk management system in Risk Dimensions. This must not be seen as a weakness of Risk Dimensions, but rather a strength. This enables the software program to calculate risk measures for portfolios that are far more complex than the portfolio in the case study.

It is also important to note that not all the capabilities of Risk Dimensions were discussed in this document. Risk Dimensions still has advanced features such as copulas and C functions, which may be used in the creation of a risk management system in other case studies.

It is also relatively easy to update the risk management system. Usually, only the market and position information that change, have to be updated. The time that Risk Dimensions takes to execute the risk management system is also relatively quick.

Thus, to conclude it can be stated without any doubt that SAS Risk Dimensions is a very powerful software program that may be used with great success in any risk management environment.

REFERENCES

DOBSON, A.J. (1990). *An introduction to Generalized Linear Models*. London: Chapman and Hall.

HULL, J.C. (2003). *Options, Futures, & Other Derivatives*. New Jersey: Prentice Hall.

JOHNSON, R.A. and Wichern, D.W. (2002). *Applied Multivariate Statistical Anlaysis*. New Jersey: Prentice Hall.

SAS INSTITUTE INC. (2000). *Risk Dimensions Administration and Configuration, Release 3.1*. Cary, NC: SAS Institute Inc.

SAS INSTITUTE INC. (2000). *Risk Dimensions Reference: Analysis and Modelling, Release 3.1*. Cary, NC: SAS Institute Inc.

SAS INSTITUTE INC. (2000). *Risk Dimensions Reference: Graphical User Interface (GUI), Release 3.1*. Cary, NC: SAS Institute Inc.
GUI,

SAS INSTITUTE INC. (2002). *Risk Dimensions: Configuration and Analysis Course Notes*. Cary, NC: SAS Institute Inc.

SAS INSTITUTE INC. (2002). *Risk Dimensions: Configuration and Analysis Demonstrations, Exercised and Solutions Handout*. Cary, NC: SAS Institute Inc.

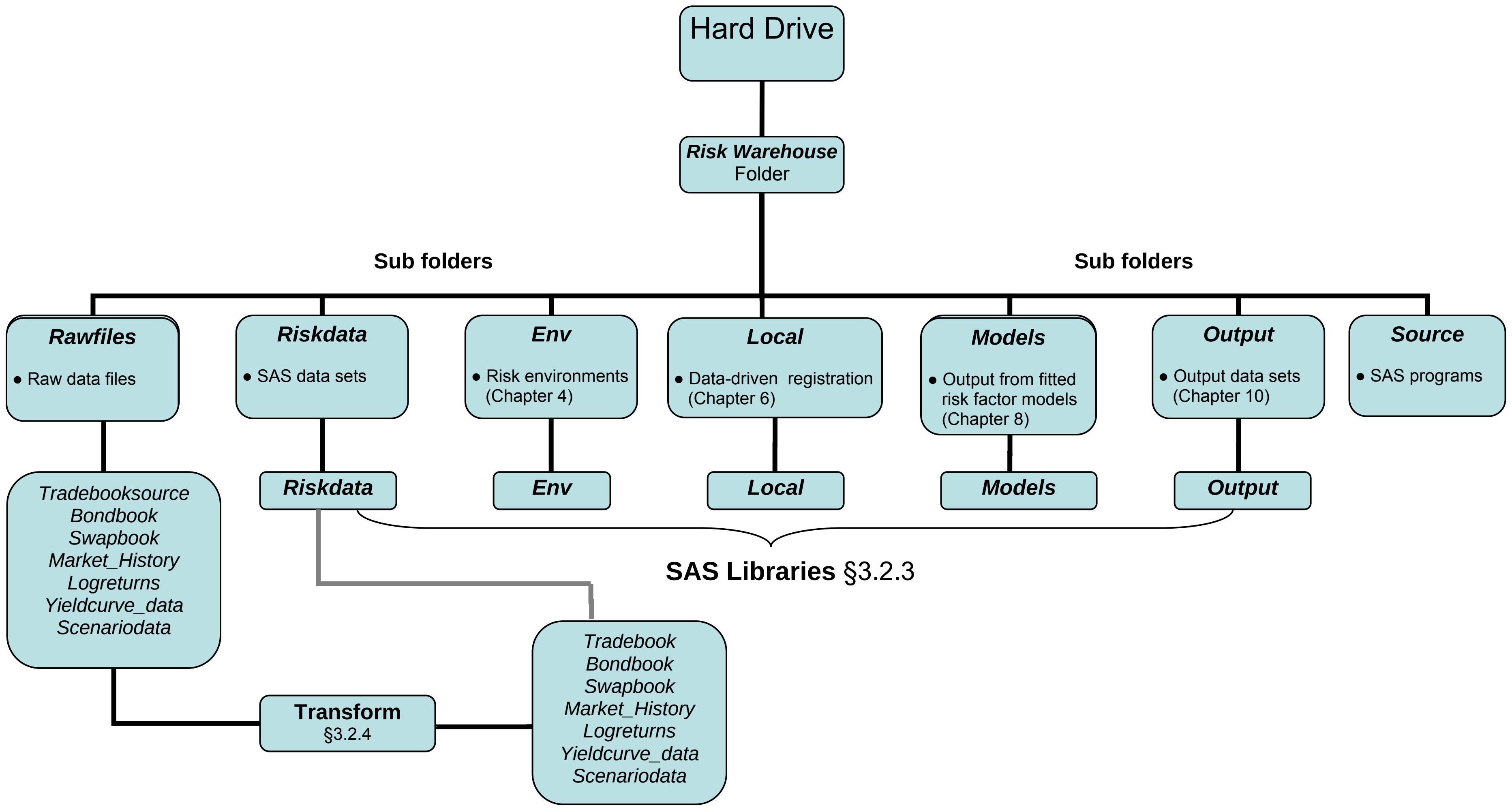
SAS INSTITUTE INC. (2002). *SAS Essentials for Risk Dimensions Course Notes*. Cary, NC: SAS Institute Inc.

SAS INSTITUTE INC. (2004). *Risk Dimensions: Configuration and Analysis Course Notes*. Cary, NC: SAS Institute Inc.

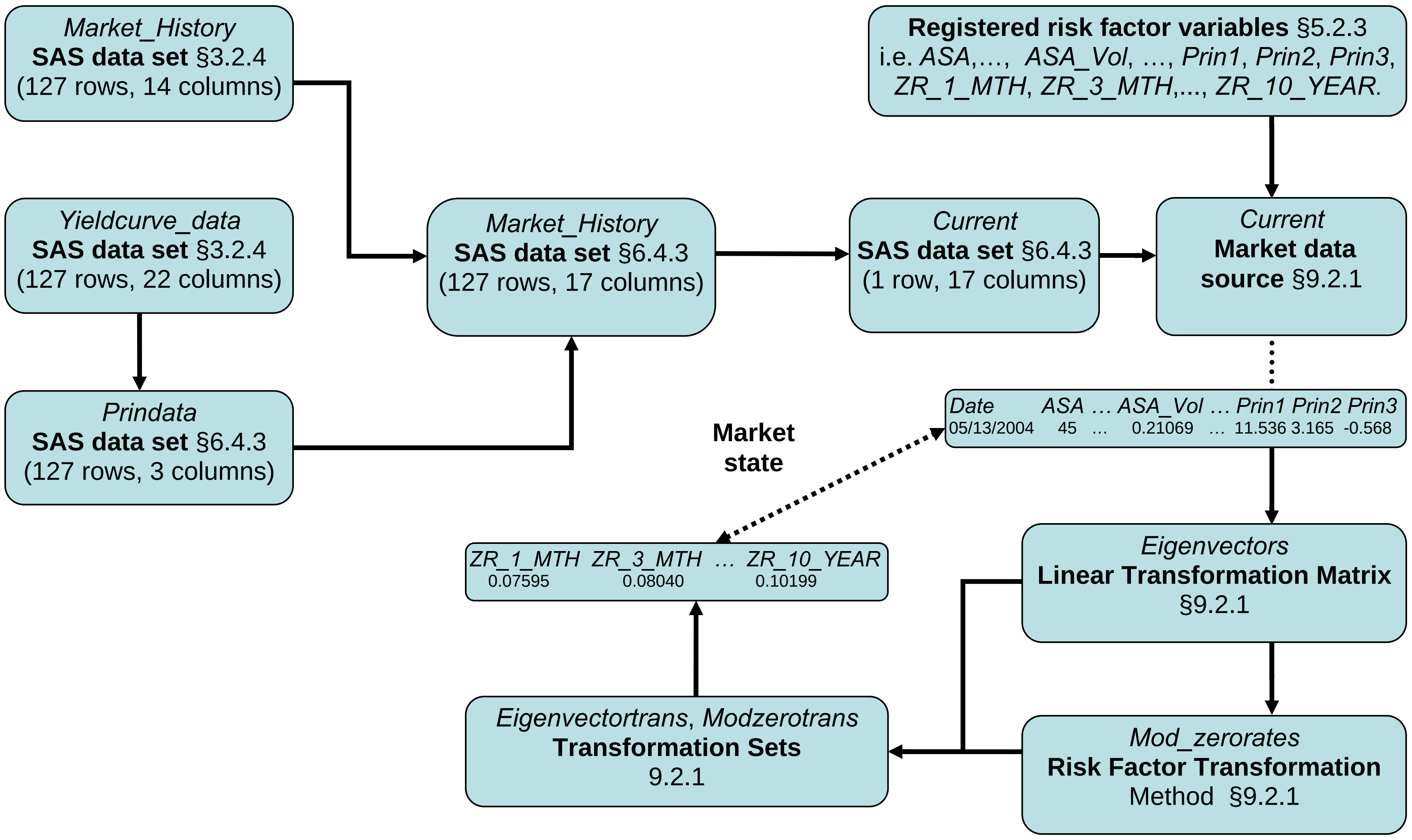
APPENDIX

The role that some of the most important Risk Dimensions structures play in the risk management program is graphically illustrated on the next few pages.

A1: Workspace on hard drive

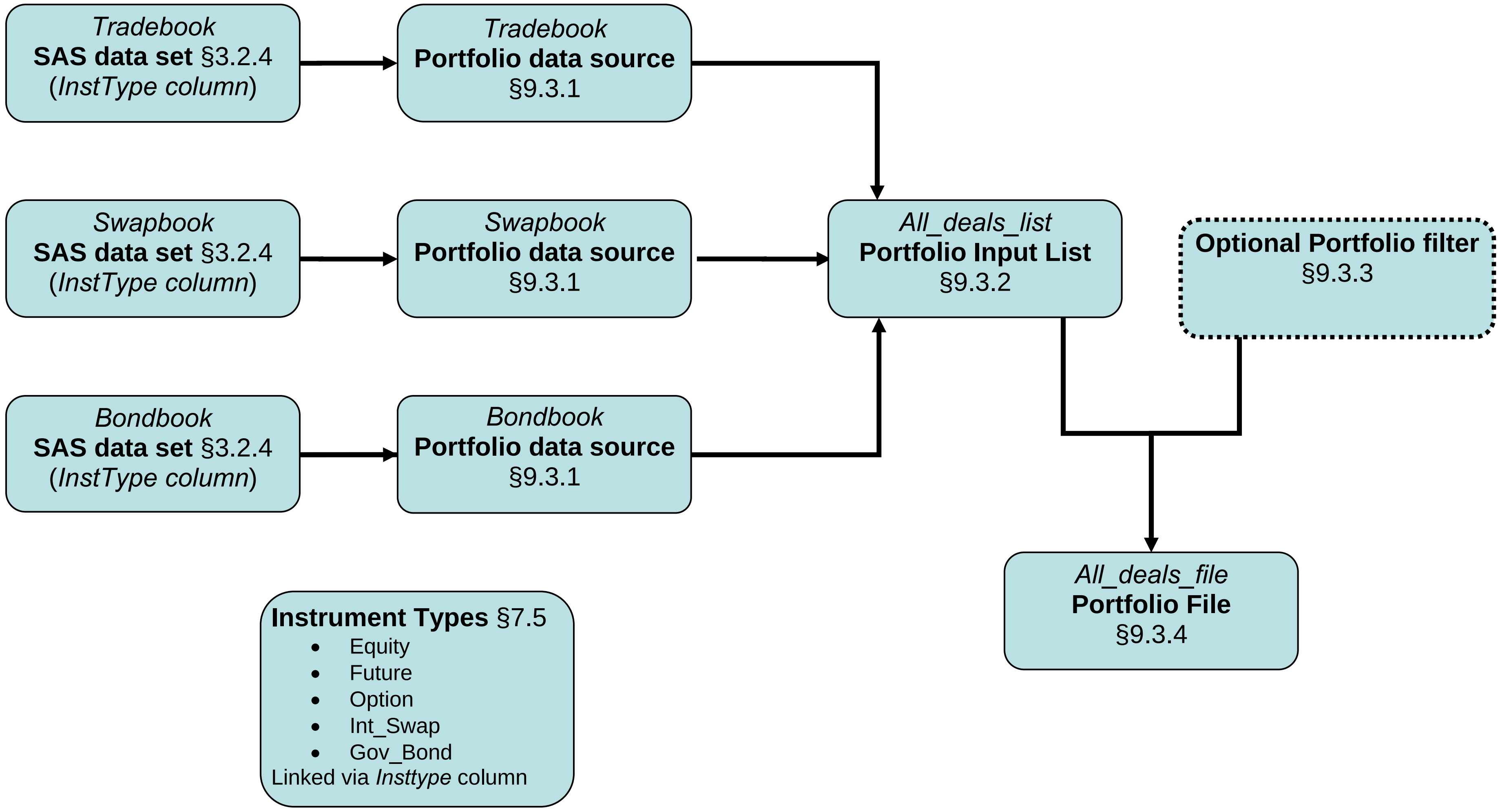


A2: The creation of the base case market state

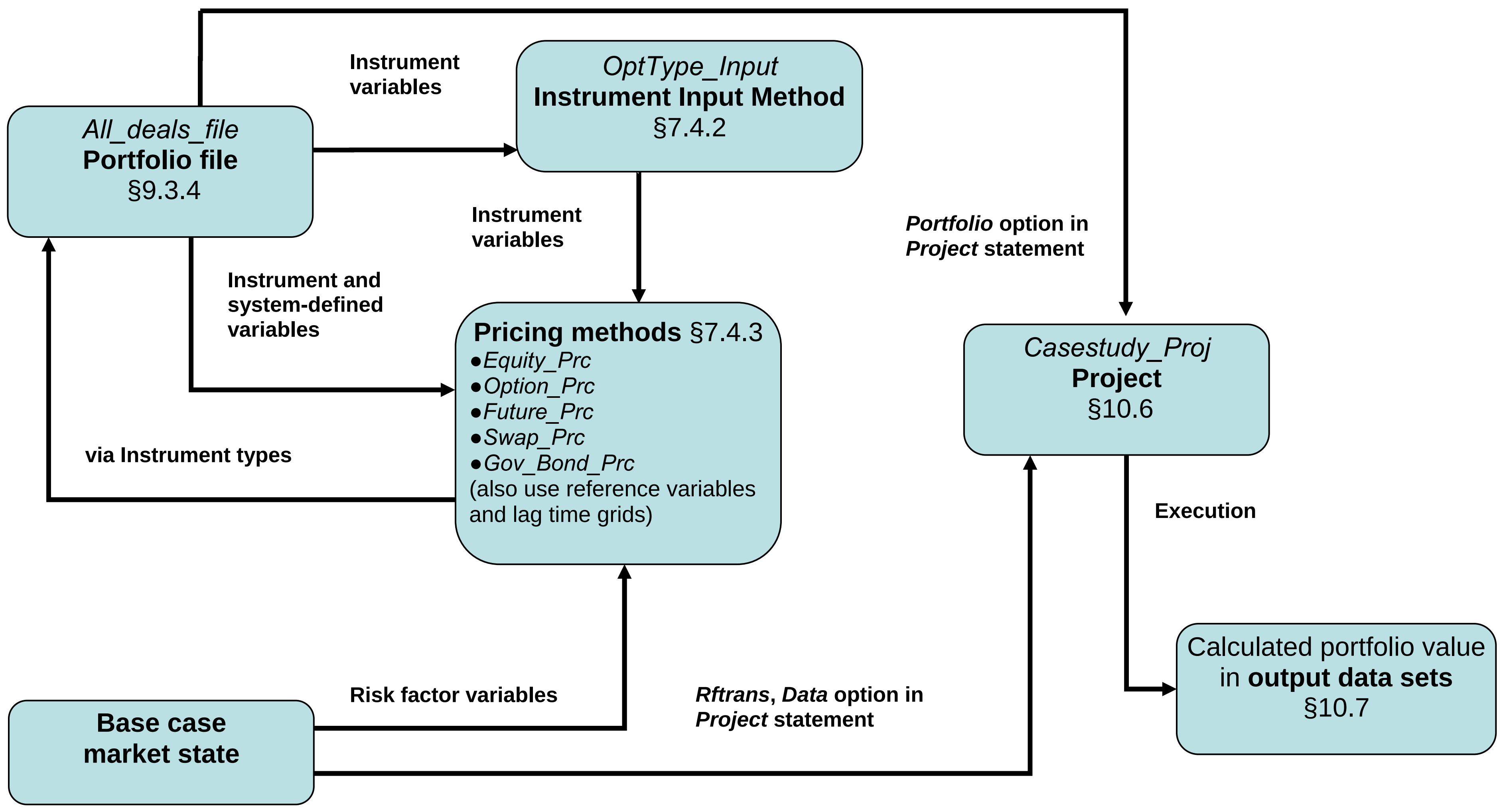


Market_History **SAS data set** §3.2.4 (127 rows, 14 columns)
Yieldcurve_data **SAS data set** §3.2.4 (127 rows, 22 columns)
Prindata **SAS data set** §6.4.3 (127 rows, 3 columns)
Market_History **SAS data set** §6.4.3 (127 rows, 17 columns)
Current **SAS data set** §6.4.3 (1 row, 17 columns)
Registered risk factor variables §5.2.3 i.e. *ASA*,…, *ASA_Vol*, …, *Prin1*, *Prin2*, *Prin3*, *ZR_1_MTH*, *ZR_3_MTH*,..., *ZR_10_YEAR*.
Current **Market data source** §9.2.1
Date *ASA* … *ASA_Vol* … *Prin1* *Prin2* *Prin3*
05/13/2004 45 … 0.21069 … 11.536 3.165 -0.568
Market state
ZR_1_MTH *ZR_3_MTH* … *ZR_10_YEAR*
0.07595 0.08040 0.10199
Eigenvectors **Linear Transformation Matrix** §9.2.1
Eigenvectortrans, *Modzerotrans* **Transformation Sets** 9.2.1
Mod_zerorates **Risk Factor Transformation** Method §9.2.1

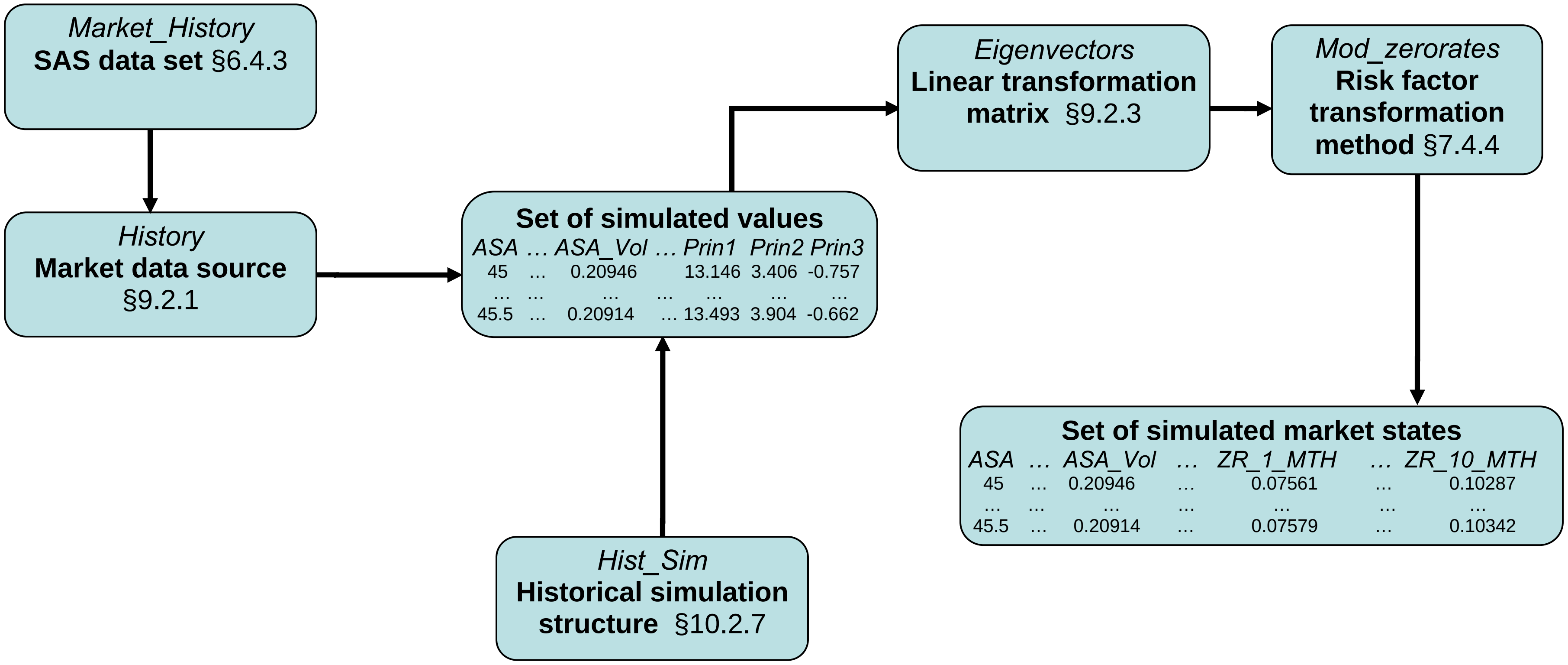
A3:The creation of the *All_deals_file* portfolio file



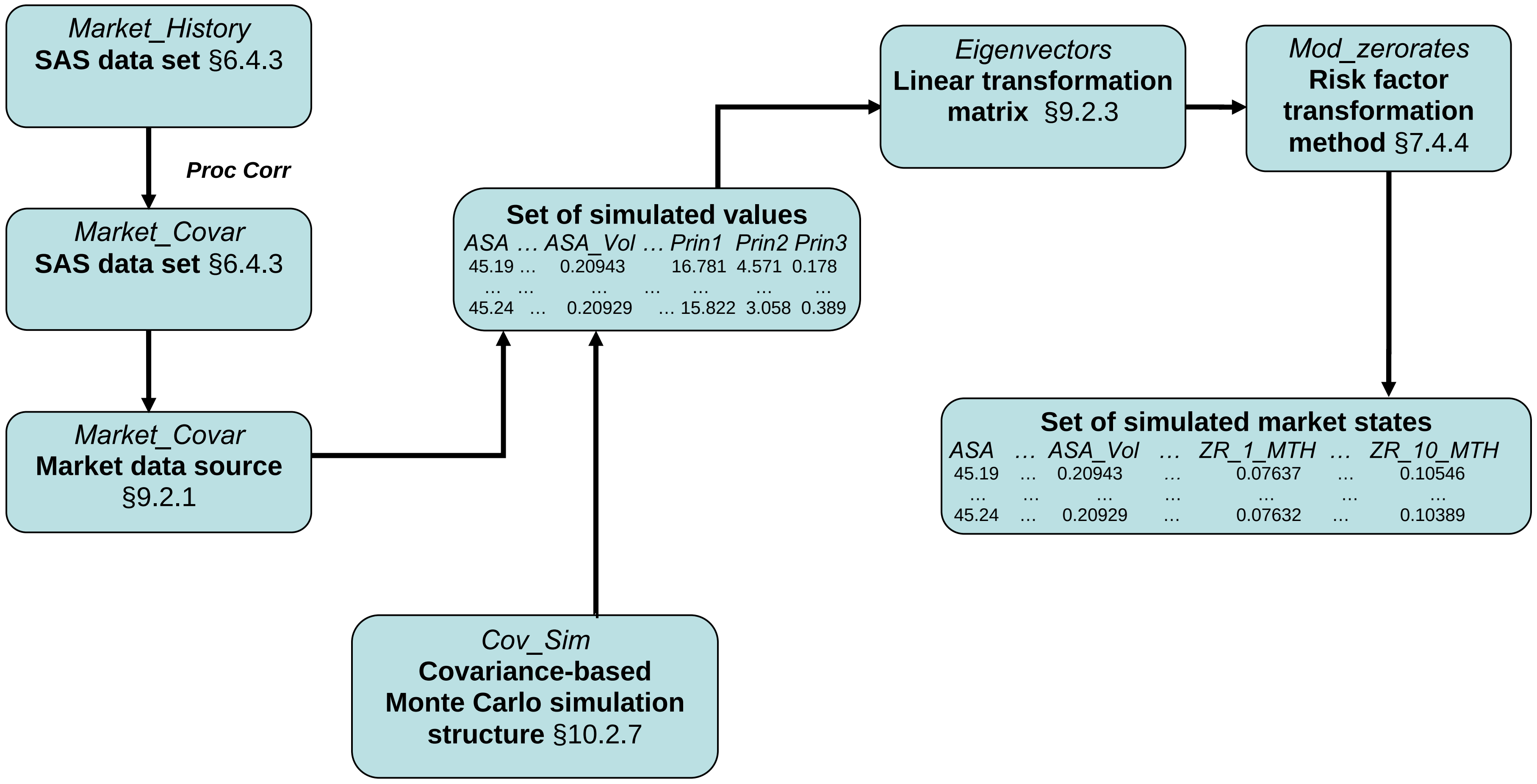
A4: The calculation of the mark-to-market value of the portfolio


All_deals_file
Portfolio file
§9.3.4
Instrument variables
OptType_Input
Instrument Input Method
§7.4.2
Instrument variables
Instrument and system-defined variables
Portfolio **option in *Project* statement**
Pricing methods §7.4.3
●*Equity_Prc*
●*Option_Prc*
●*Future_Prc*
●*Swap_Prc*
●*Gov_Bond_Prc*
(also use reference variables and lag time grids)
via Instrument types
Casestudy_Proj
Project
§10.6
Execution
Calculated portfolio value in **output data sets**
§10.7
Risk factor variables
Rftrans, ***Data*** **option in *Project* statement**
Base case market state

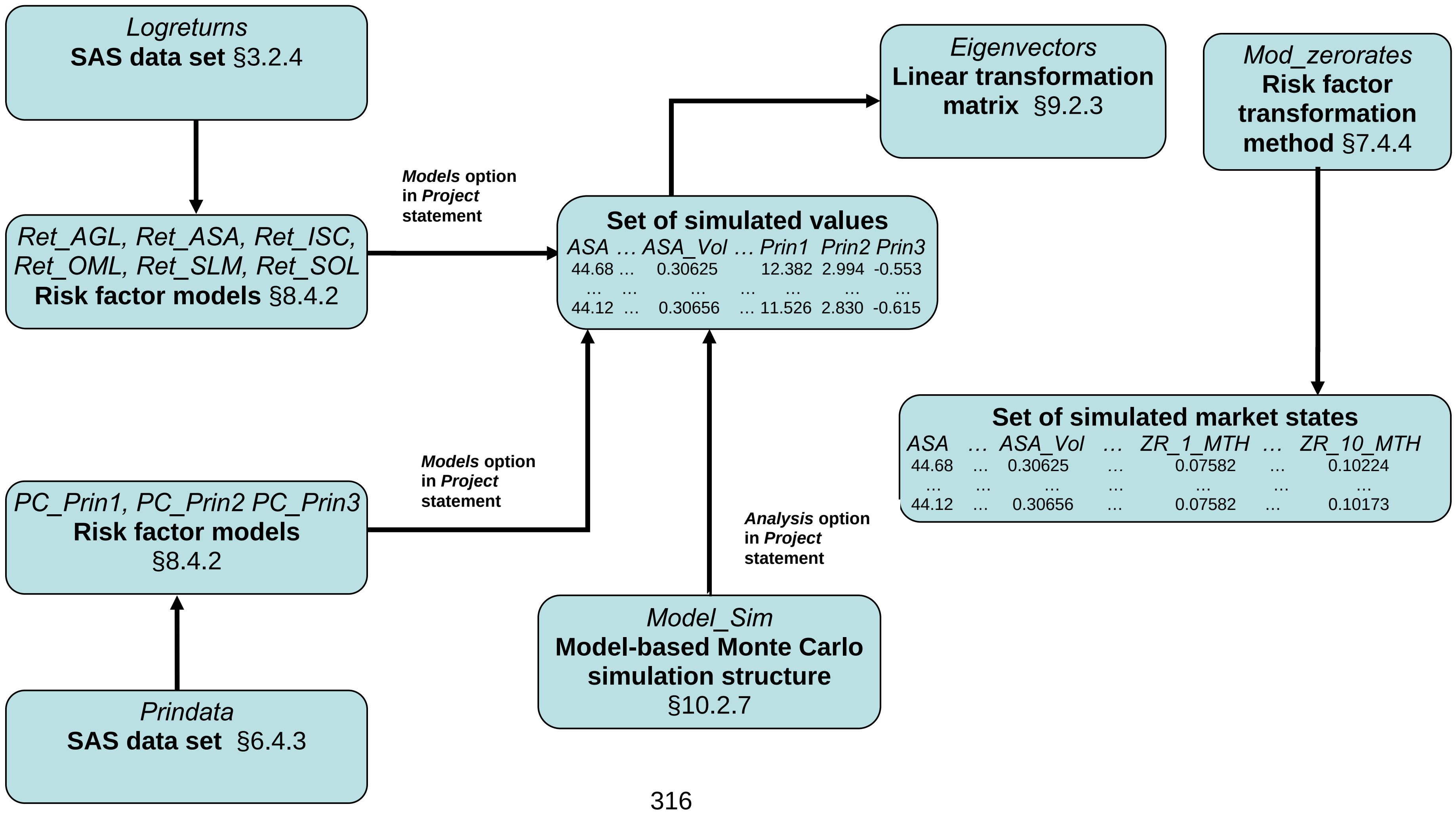
A5: Set of Simulated Market States used in the Historical Simulation VaR methodology


Market_History
SAS data set §6.4.3
History
Market data source §9.2.1
Set of simulated values
ASA … ASA_Vol … Prin1 Prin2 Prin3
45 … 0.20946 13.146 3.406 -0.757
… … … … … … …
45.5 … 0.20914 … 13.493 3.904 -0.662
Hist_Sim
Historical simulation structure §10.2.7
Eigenvectors
Linear transformation matrix §9.2.3
Mod_zerorates
Risk factor transformation method §7.4.4
Set of simulated market states
ASA … ASA_Vol … ZR_1_MTH … ZR_10_MTH
45 … 0.20946 … 0.07561 … 0.10287
… … … … … … …
45.5 … 0.20914 … 0.07579 … 0.10342

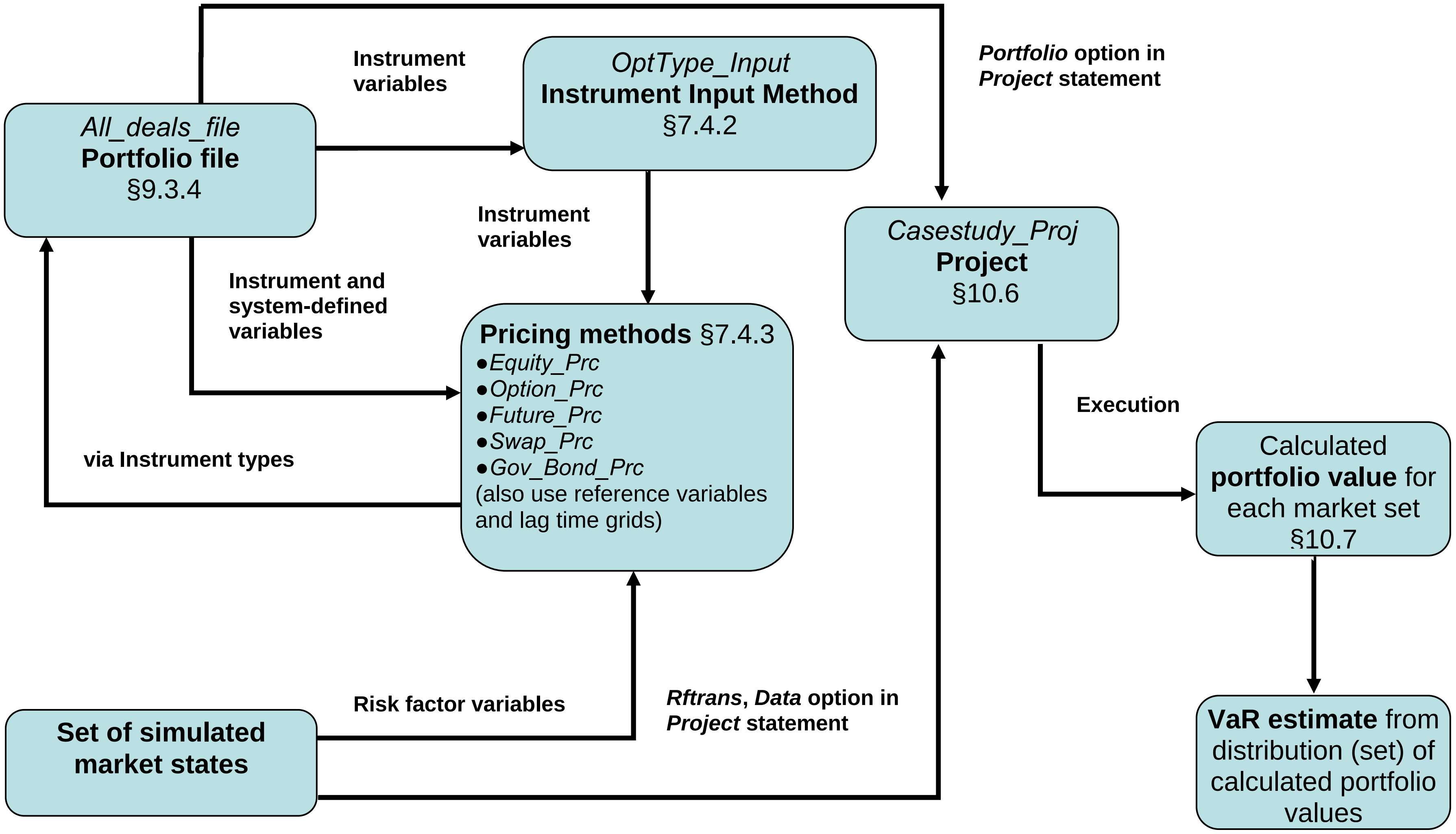
A6: Set of Simulated Market States used in the Covariance-based Monte Carlo Simulation VaR methodology



A7: Set of Simulated Market States used in the Model-based Monte Carlo Simulation VaR methodology


Logreturns
SAS data set §3.2.4
Ret_AGL, Ret_ASA, Ret_ISC, Ret_OML, Ret_SLM, Ret_SOL
Risk factor models §8.4.2
Models **option in *Project* statement**
Set of simulated values
ASA … ASA_Vol … Prin1 Prin2 Prin3
44.68 … 0.30625 12.382 2.994 -0.553
… … … … … … …
44.12 … 0.30656 … 11.526 2.830 -0.615
Eigenvectors
Linear transformation matrix §9.2.3
Mod_zerorates
Risk factor transformation method §7.4.4
Set of simulated market states
ASA … ASA_Vol … ZR_1_MTH … ZR_10_MTH
44.68 … 0.30625 … 0.07582 … 0.10224
… … … … … … …
44.12 … 0.30656 … 0.07582 … 0.10173
PC_Prin1, PC_Prin2 PC_Prin3
Risk factor models §8.4.2
Models **option in *Project* statement**
Analysis **option in *Project* statement**
Prindata
SAS data set §6.4.3
Model_Sim
Model-based Monte Carlo simulation structure §10.2.7

A8: Calculation of VaR estimates (all three methodologies)



All_deals_file
Portfolio file
§9.3.4
Instrument variables
OptType_Input
Instrument Input Method
§7.4.2
***Portfolio* option in *Project* statement**
Casestudy_Proj
Project
§10.6
Instrument variables
Instrument and system-defined variables
Pricing methods §7.4.3
●*Equity_Prc*
●*Option_Prc*
●*Future_Prc*
●*Swap_Prc*
●*Gov_Bond_Prc*
(also use reference variables and lag time grids)
via Instrument types
Execution
Calculated **portfolio value** for each market set §10.7
Risk factor variables
***Rftrans*, *Data* option in *Project* statement**
Set of simulated market states
VaR estimate from distribution (set) of calculated portfolio values