Wiley Finance Series

Hedge Fund Modelling and Analysis

An Object Oriented Approach Using C++

PAUL DARBYSHIRE
DAVID HAMPTON

WILEY

Hedge Fund Modelling and Analysis

An Object Oriented Approach Using C++

PAUL DARBYSHIRE
DAVID HAMPTON

WILEY

This edition first published 2017

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

Library of Congress Cataloging-in-Publication Data is available

A catalogue record for this book is available from the British Library.

ISBN 978-1-118-87957-3 (hbk) ISBN 978-1-118-87955-9 (ebk)
ISBN 978-1-118-87956-6 (ebk) ISBN 978-1-118-87954-2 (ebk)

Cover Design: Wiley
Cover Images: Top Image: ©iStock.com/agsandrew
Bottom Image: ©iStock.com/Storman

Set in 11/13pt Times by Aptara Inc., New Delhi, India
Printed in Great Britain by TJ International Ltd, Padstow, Cornwall, UK

Mum and Dad,

Whose love and support encourages me to achieve success.
– P.D.

For Marie-Christine, Juliette and Antoine.
– D.H.

Preface

This book is a practical introduction to modelling and analysing hedge funds using the C++ programming language. The structure of the book is as follows. Chapter 1 gives an overview of the C++ syntax in enough detail to approach the material covered in the technical chapters. Chapter 1 also introduces the concept of object oriented programming which allow us to build large and complex programs that can be broken down into smaller self-contained reusable code units known as classes. We will develop a series of classes throughout the book to tackle many of the problems encountered. Please note that this book is not intended to be an exhaustive exploration of C++ to solve problems in modelling and analysing hedge fund data. In addition, C++ is used to facilitate the solution of such problems through object oriented programming methods and various details highlighted as and when necessary.

Chapters 2 and 3 give an update of the current state of the global hedge fund industry and a detailed look at the primary data sources available to hedge fund managers and analysts. With this fundamental knowledge in place, Chapters 4–7 cover the more quantitative and theoretical material needed to effectively analyse a series of hedge fund returns and extract the relevant information required in order to make critical investment decisions.

C++ SOURCE CODE

Throughout the book there are numerous C++ source boxes (e.g., Source 2.4) typically listing the `AClass.h`, `AClass.cpp`, and `main.cpp` files and a console window showing the results of the class implementation. For example, an extract from the `Optimise` class is shown in Source P.1.

SOURCE P.1: A SAMPLE C++ SOURCE CODE

```
// Optimise.h
#pragma once;
```

```
#include "Matrix.h"
#include "Stats.h"

class Optimise: public Stats
{
   public:
    Optimise() {}
    virtual ~Optimise() {}

    // Member function declarations
    Matrix PRet(const V2DD& v); // PRet()
    Matrix PVar(const V2DD& v); // PVar()
   private:
    // Member variable declarations
    Matrix m_matrix; // An instance of the Matrix class
};

// Optimise.cpp
#include "Optimise.h"

Matrix Optimise::PRet(const V2DD& v)
{
UINT n = v[0].size()-1;

   // Declare wT and R matrices
Matrix wT = Matrix(1, n);
Matrix R = Matrix(n, 1);

// Transpose weights
for (UINT i=1; i<=n; i++)
   wT(1, i) = 1 / (DBL)n; // Equal weights

// Mean returns
V1DD r = Mean(v, 12);

// R matrix
   for (UINT i=1; i<=n; i++)
   R(i, 1) = r[i-1];

   return wT * R;
}

Matrix Optimise::PVar(const V2DD& v)
{
```

```
UINT n = v[0].size()-1;

// Declare w, wT and VCV matrices
Matrix w = Matrix(n, 1);
Matrix wT = Matrix(1, n);
Matrix VCV = Matrix(n, n);

   // Initialise portfolio weights
for (int i=1; i<=n; i++)
    w(i, 1) = 1 / (DBL)n; // Equal weights

// Transpose weights
   for (int i=1; i<=n; i++)
    wT(1, i) = 1 / (DBL)n; // Equal weights

// Covariance matrix
V1DD cov = Cov(v);

// VCV matrix
int k = 0; // Covariance offset
for (UINT i=1; i<=n; i++)
{
  for (UINT j=1; j<=n; j++)
  {
     VCV(i, j) = cov[j+k-1];
      if(i == j)
        VCV(i, j) *= 12; // Annualise variance
  }
  k+=10;
}

return wT * VCV * w;
}

// ...
// main.cpp
// ...

// Create class instances
Import thfs;
Optimise optimise;

// Declare and call GetData()
V2DD data = thfs.GetData("./data/10_hedge_funds.dat");
```

```
// Declare and call PRet() and PVar() member function
Matrix pret = optimise.PRet(data);
Matrix pvar = optimise.PVar(data);

// Output results
cout << "\n Port. Ret. (%) = ";
pret.Print();
cout << " Port. Var. (%) = ";
pvar.Print();

// ...
```

Console Output

```
File:./data/10_hedge_funds.dat
Imported 720 values successfully!

Port. Ret. (%) = 8.552
Port. Var. (%) = 10.725
```

Comment blocks, such as:

```
// ...
// main.cpp
// ...
```

are used to omit parts of the source code (above and below) when new code is added to existing definitions or implementations. As we progress through the book we will gradually reduce unnecessary overuse of comments (//) within source listings once we feel confident we have clearly defined such routines and concepts in previous listings.

Please note that we do not give any warranty for completeness, nor do we guarantee that the code is error free. Any damage or loss incurred in the application of the C++ source code, algorithms and classes discussed in the book are entirely the

TABLE P.1 10 Hypothetical Hedge Funds

Hedge Fund	Abbreviation
Commodity Trading Advisor	CTA1, CTA2, CTA3
Long Short Equity	LS1, LS2, LS3
Global Macro	GM1, GM2
Market Neutral	MN1, MN2

reader's responsibility. If you notice any errors in the C++ source code, algorithms or classes, or you wish to submit some new method as a C++ function, algorithm, class, model or some improvement of the method illustrated in the book, you are very welcome.

HYPOTHETICAL HEDGE FUND DATA

Throughout the book there is constant reference to many monthly hedge fund return series. The 10 hedge funds are all *hypothetical* and have been simulated by the authors as a unique data set for demonstration purposes only. The techniques and models used in the book can therefore be tested on the hypothetical data before being applied to real-life situations by the reader. The hypothetical data is nonetheless close to what would be expected in reality. The 10 funds are a mixture of several major hedge fund strategies i.e. *Commodity Trading Advisor* (CTA), *Long/Short Equity* (LS), *Global Macro* (GM) and *Market Neutral* (MN) strategies as described in Table P.1.

All data files used throughout the book are identified in italics e.g. *10_hedge_funds.dat*.

BOOK WEBSITE

The official website for the book is located at: www.darbyshirehampton.com

The website provides free downloads to all of the hypothetical data, C++ programs and classes, as well as many other useful resources.

The authors can be contacted on any matter relating to the book, or in a professional capacity, at the following email addresses:

Paul Darbyshire: pd@darbyshirehampton.com

David Hampton: dh@darbyshirehampton.com

Contents

1

Essential C++

This chapter covers the fundamental requirements necessary to allow the reader to get up and running building quantitative models using the C++ programming language. This introduction is in no way intended to be an in-depth treatment of the C++ programming language but more an overview of the basics required to build your own efficient and adaptable programs. Once the key concepts have been developed, object-oriented principles are introduced and many of the advantages of building quantitative systems using such programming approaches are outlined. It is assumed that the reader will have some prerequisite knowledge of a low-level programming language and the necessary computation skills to effectively grasp and apply the material presented here.

1.1 A BRIEF HISTORY OF C AND C++

C is a *procedural*[1] programming language developed at Bell Laboratories between 1969 and 1973 for the UNIX operating system. Early versions of C were known as K&R C after the publication of the book *The C Programming Language* written by Brian Kernighan and Dennis Ritchie in 1978. However, as the language developed and became more standardised, a version known as ANSI[2] C became more prominent. Although C is no longer the choice of many developers, there is still a huge amount of *legacy* software coded in it that is actively maintained. Indeed, C has greatly influenced other programming languages, in particular C++ which began purely as an extension of C.

[1] *Procedural* programming is a form of *imperative programming* in which a program is built from one or more procedures i.e. subroutines or functions.

[2] Founded in 1918, the *American National Standards Institute* (ANSI) is a private, non-profit membership organisation that facilitates the development of *American National Standards* (ANS) by accrediting the procedures of the *Standards Developing Organizations* (SDOs). These groups work cooperatively to develop voluntary national consensus standards.

Often described as a *superset* of the C language, C++ uses an entirely different set of programming concepts designed around the *Object-Oriented Programming* (OOP) paradigm. Solving a computer problem with OOP involves the design of so-called *classes* that are abstractions of physical objects containing the state, members, capabilities and methods of the object. C++ was initially developed by Bjarne Stroustrup in 1979 whilst at Bell Laboratories as an enhancement to C; originally known as C with Classes. The language was renamed C++ in the early 80s and by 1998, C++ was standardised as ANSI/ISO[3] C++. During this time several new features were added to the language, including virtual functions, operator overloading, multiple inheritance and exception handling. The ANSI/ISO standard is based on two main components: the *core language* and the *C++ Standard Library* that incorporates the C Standard Library with a number of modifications optimised for use with the C++ language. The C++ Standard Library also includes most of the *Standard Template Library* (STL); a set of tools, such as *containers* and *iterators* that provide array-like functionality, as well as *algorithms* designed specifically for sorting and searching tasks. C++11 is the most recent *complete* overhaul of the C++ programming language approved by ANSI/ISO on 12 August 2011, replacing C++03, and superseded by C++14 on 18 August 2014. The naming convention follows the tradition of naming language versions by the year of the specification's publication, although it was formerly known as C++0x to take into account many publication delays. C++14 is the informal name for the most recent revision of the C++ ANSI/ISO standard, intended to be a small extension over C++11, featuring mainly bug fixes and small syntax improvements.

1.2 A BASIC C++ PROGRAM

Without doubt the best method of learning a programming language is to actually start by writing and analysing programs. Source 1.1 implements a basic C++ program that simply outputs a string of text, once the program has been compiled and executed, to the console window. Although the program looks very simple it nevertheless contains many of the fundamental components that every C++ program generally requires.

SOURCE 1.1: A BASIC C++ PROGRAM

```
// main.cpp
 #include <windows.h>
 #include <iostream>
```

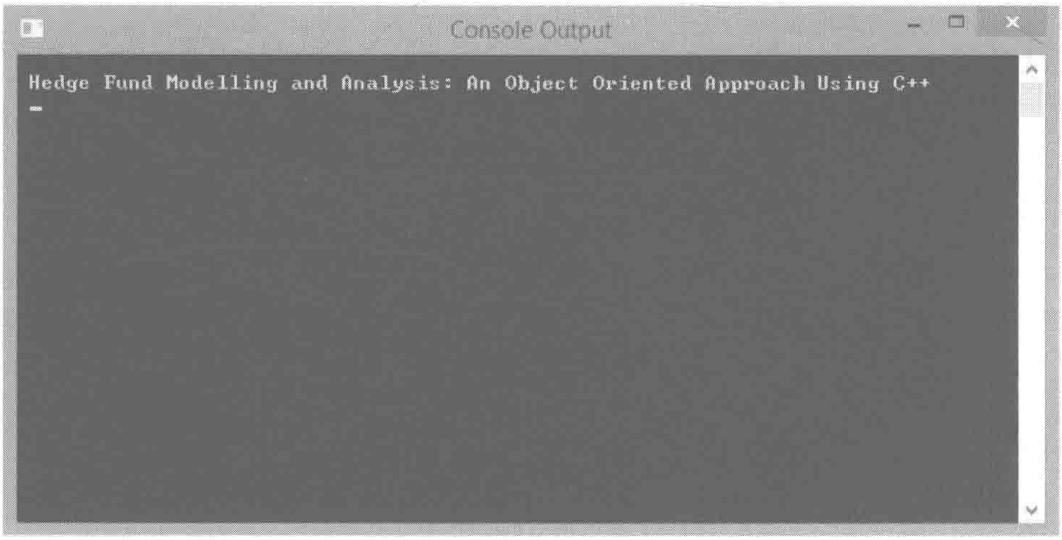
[3] The *International Organisation for Standardisation* (ISO) is an international standard-setting body made up of representatives from a range of *National Standards Organisations* (NSOs).

```
using std::cout;
using std::cin;

int main()
{
  SetConsoleTitle(L"Console Output"); // Set title of console
  window

  cout << "\n " << "Hedge Fund Modelling and Analysis: An Object
  Oriented Approach Using C++";

  cin.get(); // Pause console window
  return 0; // Return null integer and exit
}
```



Statements beginning with a hash symbol (#) indicate *directives* to the *preprocessor* that initialise when the compiler is first invoked, in this case, to inform the compiler that certain functions from the C++ Standard Library must be included. #include <windows.h> gives the program access to certain functions in the library, such as SetConsoleTitle() whilst #include <iostream> enables console input and output (I/O). Typical objects in the iostream library include cin and cout which are explicitly included through the using statement at the top of the program. Writing using std::cout at the top of the program avoids the need to keep retyping std through the *scope resolution operator* (::) every time cout is used. For example, if we had not specified using std::cout we would have to explicitly write std in front of each usage throughout the program, that is:

```
std::cout << "\n " << "Hedge Fund Modelling and Analysis: An Object
Oriented Approach Using C++";
std::cin.get();
```

Although in this case there are only two occasions where we need `std`, you can imagine how this could quickly clog up code for very large programs. Note also that all C++ statements must end with a semi-colon (`;`).

A commonly identified problem with the C language is the issue of running out of names for definitions and functions when programs reach very large sizes eventually resulting in name clashes. Standard C++ has a mechanism to prevent such a clash through the use of the `namespace` keyword. Each set of C++ definitions in a library or program is *wrapped* into a namespace, and if some other definition has an identical name, but is in a different namespace, then there is *no* conflict. All Standard C++ libraries are wrapped in a single namespace called `std` and invoked with the `using` keyword:

```
using namespace std;
```

Whether to use `using namespace std` or explicitly state their use through `using std::cout`, for example, is purely a preference of programming style. The main reason we do not invoke `using namespace std` in our programs is that this leaves us the opportunity of defining our own namespaces if we wish and it is generally good practice to have only one namespace invocation in each program.

The `main()` function is the point at which all C++ programs start their execution even if there are several other functions declared in the same program. For this reason, it is an essential requirement that all C++ programs have a `main()` function within the body at some point in the program. Once the text is output to the console window, `cin.get()` is used to cause the program to pause so that the user can read the output and then close and exit the window by pressing any key. Technically, in C or C++ the `main()` function must return a value because it is declared as `int` i.e. the main function should return an integer data type. The `int` value that `main()` returns is usually the value that will be passed back to the operating system; in this case it is `0` i.e. `return 0` which indicates that the program ran successfully. It is not necessary to state `return 0` explicitly, because the compiler invokes this automatically when `main()` terminates, but it is good practice to include a return type for all functions (including `main()`).

1.3 VARIABLES

A variable is a name associated with a portion of memory used to *store* and *manipulate* the data associated with that variable. The compiler sets aside a specific amount of memory space to store the data *assigned* to the variable and associates the variable name with that memory *address*. As the name implies, variables can be changed within a program as and when required. When new data is assigned to the same variable, the old data is *overwritten* and restored in the same memory address. The data stored in a