Course Computational and Quantitative Finance in C++

Contents

Module 1: Primer and Fundamental C++ Syntax
In this module we introduce basics of the C++ language, including essential syntax, how to create functions and classes and how to integrate the code into a C++ project.

- Refresher C language
- Learning the C++ project environment
- From source code to running program
- Creating basic C++ classes: header and code files
- Creating robust classes (const, call by value/reference)
- Operator overloading in C++
- Creating user-defined operators
- Memory management: heap, stack and static
- Implementing contracts: exception handling in C++
- Project: creation simple C++ classes for financial derivatives

After having completed this module you will be in a position to write, compile and run C++ applications and be able to test and debug code quickly and effectively. This means that you will not lose valuable time. We take a number of examples from finance, namely exact formulas for option pricing and the creation of C++ classes that model derivatives.

Module 2: Advanced C++
In this module we introduce a number of advanced techniques that promote the flexibility and robustness of your C++ applications. This is a crucial module because many C++ applications use these techniques and they allow us to extend and modify system code with a minimum of impact on the stability of the application.

- Pointers: native C++, C++0X pointers
- Modelling functions: by pointers and by function objects
- Applications in finance
- An introduction to inheritance and composition in C++
- Virtual and pure virtual functions
- Tips and guidelines when using inheritance
- Combining inheritance and composition
- Run-Time Type Information (RTTI)
- Factoring common code using the Template Method Pattern
- Project: creating flexible payoff hierarchies

After having completed this module you will be able to create extendible and understandable C++ class hierarchies for financial derivatives. We shall use these classes as reusable building blocks when we develop applications in later modules.

Module 3: C++ Templates and the Standard Template Library (STL)
This module introduces the student to Generic Programming (GP) and its implementation in C++, namely the template mechanism. We discuss the fundamental syntax issues and we show how to create templated functions and classes. Furthermore, we show how to integrate and combine templates with the inheritance and composition techniques that we discussed in previous modules. Having learned what templates are we then proceed to discussing the most important components of STL and their applications.

- An introduction to the generic programming model
- C++ templates: functions and classes Template specialization
- Combining templates with inheritance and composition
- An overview of STL
- STL sequence containers: list, vector, deque
- STL iterators
- Associative containers: map, set, multimap, multiset
- STL algorithms: searching, sorting, extraction
- Mutating and Nonmutating algorithms
- Modifying and Nonmodifying algorithms

- Project: using templates for financial applications

After having completed this module the student will understand template programming in C++.

Module 4: Design Patterns

In this module we introduce a number of design techniques that we deploy in C++ so that our applications can be customized when requirements change (as they inevitably do). In particular, we give an overview of the famous Design Patterns (23 in total) and we apply the most important ones to examples and applications in finance.

- What is software design?
- Quick overview of the Unified Modeling Language (UML)
- The Gamma ("Gang of Four" classification
- Creational patterns: Factory, Singleton, Builder, Prototype
- Structural patterns: Bridge, Composite, Facade, Proxy
- Behavioural patterns I: Template method, Strategy, Observer
- Behavioural patterns II: Visitor, Command, Mediator
- Applying design patterns in finance: the steps
- Project: designing and implementing FDM for Black-Scholes PDE

After having completed this module you will be able to discover and apply the most appropriate design patterns for a given problem in finance.

Module 5: Libraries and Interfacing Issues

Whereas the code in Module 4 was concerned with application logic and algorithms this module discusses a number of features and tools that allow us to develop fully-fledged applications, in particular the input, processing and output modules in an application.

- C++ Excel integration: xll, Automation and COM Addins
- Creating xll applications
- Automation Addins and worksheet functions
- COM Addins
- Registration, activation, libraries
- An introduction to ATL (Active Template Library)
- Overview of the Boost library
- Boost random number generators
- Boost multi-array and property map libraries
- Introduction to XML
- DLLs and Libs
- Calibration
- Project: developing Excel Addins for Monte Carlo and Fixed Income Applications

After having completed this module you will be able to integrate your code with a number of standard software environments, such as Excel, Boost and XML.

Module 6: Integration and Applications: Overview

This is the last module of the course and it is here that we create a fully-fledged application using the experience that we have gained in the first five modules. You can choose the kind of application (equity, fixed income, commodity) and the numerical technique (FDM, Monte Carlo, ...) you wish to use.

- Analysis and system decomposition
- Defining inter-system interfaces
- Applying the GOF patterns
- An introduction to multi-threading and parallel programming
- Implementing finance applications in C++ with OpenMP
- Testing and profiling your application
- Integration with Excel
- Equity, interest rate and other applications in finance
- Monte Carlo, FDM, quadrature and lattice solutions

Module 7: The Monte Carlo Method in C++

Stochastic Differential Equations (SDE)

- Geometric Brownian Motion (GBM)
- CEV model
- Stochastic volatility

Finite Difference Method for SDE

- Euler and Milstein method for GBM
- Predictor-corrector method
- QE method

Examples

- Short-rate
- Heston
- Jump models

Monte Carlo Engine in C++

- Modular decomposition
- Design of engine (Produce-consumer)
- Random number generators
- Parallel programming

Module 8: The Finite Difference Method in C++

Finite Difference Method (FDM)

- One-factor models
- Plain and barrier options
- Early exercise features
- The Crank Nicolson method
- Comparing FDM with trinomial method

Alternating Direction Explicit (ADE) Method

- Background and motivation
- ADE for one-factor models
- ADE for nonlinear pricing models
- Advantages of ADE

Two-Factor Model

- ADI and Splitting Methods
- Craig-Sneyd method
- Mixed derivatives and Janenko method
- ADE for two-factor problems

Module 9: Interest Rate Models in C++

Overview of Bond and Fixed Income Pricing

- Bond Pricing: Design, Implementation and Excel Interfacing
- Overview of bonds and kinds of bonds
- Bond price and bond yield
- Convexity
- (Macauley) duration
- Accrued interest and dirty price
- Day count conventions

Short-term Interest Rate Futures and Options

- Introduction (short term interest rate futures and option description)
- Organizing and manage futures data and code
- Conventions for Liffe Futures
- Pricing Option
- Working Example: portfolio of options

Interest Rate Models

- Vasicek, CIR, Hull-White
- Exact solutions
- Approximate solutions: lattice, PDE, MC
- Calibration

Module 10: Excel Interoperability

- Creating Automation Add-ins in C++
- Guid, ProgId, ClassInterface
- Referencing the Excel Application
- Registering COM components
- Loading and using Automation Add-ins
- Versioning
- Volatile Cells

COM Add-ins

- Background
- ATL projects with IDTExtensibility2 support
- Managed and unmanaged add-ins
- VS add-ins and shared add-ins
- Shared Add-in Wizard
- Extendibility projects

Examples

- Monte Carlo Engine
- PDE solvers
- Integration with boost Math Toolkit

After having completed this module you will have used C++ in combination with mathematical methods for finance to produce a working system.

Project: term (final) project

The examiners will review your project and a small exam will be given.