MF 703: Programming for Mathematical Finance

Professor Chris Kelliher

Boston University Questrom School of Business

Office hours: Monday 10:30-11:30am ET via Zoom

Email: cmk44@bu.edu

Teaching Assistants:

- George Kepertis. Email: kepertis@bu.edu
  Office hours: Tuesday 11am-12pm
- Nikos Vingos. Email: nvingos@bu.edu
  Office hours: Wednesday 12-1pm

About the Course

- What this course is...
  - A high-level programming course that will focus almost exclusively on financial applications.
  - A place to learn how to write good, safe, scalable code.
  - It should also help you avoid many common coding pitfalls.
  - You will have a chance to implement many practical applications that I have faced repeatedly in my career and benefit from my insight into these problems.
- What this course is not...
  - A place where I will cover the basics of programming
  - A place to learn the syntax of any programming language
  - A place to dive deeply into the theory behind each financial application

Course Goals

- This course will give you an overview of the most common software engineering problems that quants face.
- It will also introduce you to many of the most common quantitative tools & methods used in mathematical finance.
- It will provide you with hands-on experience implementing these methods as well as commonly used financial models.
- Along the way I will provide context for each of the problems that we face. *Always be mindful of the strengths and weaknesses of the techniques that you learn.*
- This course will teach you how to write robust, scalable code and work on large-scale applications.
- This course will give you the tools required to work through all phases of a programming project.

Course Requirements

- This course assumes that you have taken multiple programming / computer science courses and are familiar with C++ and Python.
- This course assumes knowledge of linear algebra, stochastic calculus, statistics, and probability.
- Additionally, we assume that you have been exposed to the concepts of asset pricing and risk-neutral valuation.

My Programming Expectations

In order to take this course I expect that you:

- Have completed at least one project in Python or C++.
- Are familiar with object-oriented programming concepts and have some experience writing user defined classes, recursive functions, and working with 2-dimensional lists.
- Have completed at least one significant project with finance applications, such as one involving monte carlo simulation, binomial trees or other options pricing models.

Generally speaking my expectation is that you already know how to code, and are here to learn how to write better code, and learn how to apply it to practical financial problems.

Course Topics

In this course we will cover the following programming topics:

1. Foundations of Programming in SQL / Python / C++
   (a) Programming in a multi-programmer environment
   (b) Data Collection & Cleaning
   (c) Database Design
   (d) SQL Programming
   (e) Development in Python & C++
   (f) Principles of Object-Oriented Programming

Course Topics

And we will cover the following practical applications:

1. Algorithms and Financial Model Development
   (a) Options Pricing
   (b) Greeks & Hedging
   (c) Exotic Options
   (d) Fixed Income Models
   (e) Risk Models: VaR / CVaR
   (f) Monte Carlo Simulation
   (g) Portfolio Optimization
   (h) Algorithmic Trading

Course Logistics

- Before we dive in, let's discuss logistics (grading, homework, exams, final projects, etc.)

Questrom Covid Statement of Norms

- Use the designated paths and doors to enter and exit the classrooms and move around the Hariri building.
- Use the wipes provided in classrooms to clean desktops and other spaces before and after use.
- Wear an appropriate personal protection equipment (PPE) face covering when in the Hariri building, including in the Hariri classrooms and offices. Students not wearing a face covering will be asked to leave and are expected to comply with the request.
- Be willing to display your "green screen" compliance app upon request (e.g., in class or for a meeting).
- Eating and drinking are NOT permitted in the classroom.
- Update your LfA location and class status (in-person or 100% remote) on the StudentLink as changes occur throughout the

semester.

- Attend class remotely if your behavior outside of the classroom might put others in the community at risk.

Course Lectures

- This course will rely heavily on synchronous learning for both in-person and remote students.
- My goal is to make sure that all students have the opportunity to learn, ask questions and gain experience coding.
- Whether you are attending this course in-person or remotely, you are encouraged to participate in our discussions!
- In addition to the programming topics covered, I strive to include insights from my career and hope to make the course directly relevant to your future in industry.

Remote Attendance

- If you are attending the course remotely, you are welcome to attend the section of the course that is most convenient, especially if you are in a different timezone.
- My only request is that if you are doing this you attend the same section every week as this will facilitate my planning the lectures.

Attendance Policy

- Attending the lectures while strongly encouraged is not required.
- In an effort to create a dynamic classroom environment where students feel most comfortable asking questions, the lectures will not be recorded.
- Having said that the slides are designed to be sufficient to learning the material should you need to miss any lectures.

Collaboration

- Close collaboration on homework assignments and your final projects is highly encouraged.
- Collaboring remotely brings an added challenge but is also a new and relevant skill for each of you.
- If you are joining us remotely you will still need to find a group for your projects and find a cohesive way to operate as a team.
- Collaborating in a single code base safely is also a valuable skill, as you will see on your projects.

Course Resources

- The following documentation may be helpful as we move through the course:
  - Python
  - C++
- NOTE: If you have limited experience with either of these languages you may want to spend some time reviewing these resources before we go too deeply into applications.
- No book is required for this course as the slides, and the references that I provide should be sufficient.

Grading

- Your final grade for this course will be assigned based on your performance in:
  1. An Exam
  2. A final project
  3. Homework
- The exam will make up 35% of your final grade
- The final project will make up 35% of your final grade
- Homework will make up 30% of your final grade

Exam

- There will be a take-home exam on November 5th
- All reference materials will be allowed during the exam period, which will last 24 hours.
- Collaboration, however, is forbidden on the take-home exam.
- A practice exam will be provided prior to the exam. If time permits, we will dedicate some portion of class-time to solving the practice exam.
- As it gets closer, I will also provide more clarity on the types of questions that will asked, what topics will be emphasized, etc.

Final Project

- You will be required to build teams of 3-5 people to work on a project of practical relevance.
- You will be asked to choose a quantitative technique and build an accompanying piece of software.
- Your team will be asked to give a 10-minute demo of your software during the last day of class.
- In addition, your team will be required to submit a written summary of at most 10 pages containing your method and results.
- NOTE: All group members to contribute to their team's final project. We also expect all group members to be able to discuss all aspect's of their teams work. We reserve the right to ask any group or individual member about any aspect of their project and incorporate these answers into the final project grade.

Final Project

- For your final project we expect you to:
  1. Choose a topic/technique directly related to quantitative finance.
  2. Research this topic/technique thoroughly, highlighting its applications in finance and potential uses.
  3. Apply this topic to a real-world problem. This includes implementing the topic/technique in C++ or Python and obtaining substantive results.
  4. Analyze the strengths and weaknesses of the topic/technique based on your results as well as external research.
  5. NOTE: It is not expected that you do ground-breaking research on this project; however, it is expected that you provide a robust implementation of a technique that interests you.

Final Project Proposal

- You will be required to choose a topic and submit a project proposal to me by October 15th.
- I will give you a significant amount of flexibility in choosing your topic for the final project; however, it must be directly related to the topics that we discuss in class.
- Your proposal should describe the topic that you are planning to research, and include at least 3 references that you plan to leverage in your research.

Final Project Topics

- I encourage you to discuss your topic with me prior to the proposal deadline.
- I am happy to provide some examples of project topics in class prior to the deadline, but I also want to hear from you what your interests are.

Final Project Grading

- As this is a programming course, a significant part of the grade for your final project will be based on the quality of your implementation.
- Remember to use and apply all the principles that we cover in class on your projects.
  - As an example, all projects should contain evidence of unit tests and other validation exercises to show that the model was implemented correctly.

Homework

- Homework will be assigned every week.
- The assignments will require a significant amount of coding, which should be written in Python or C++. Unless otherwise specified on the assignment, you can choose which of the two languages to use.
- Students are encouraged to discuss their homework assignments; however, all solutions and any code must be unique and written by the student submitting the assignment.
- Working with other students is a great way to debug your code, but please, make sure that each piece of code you submit was written by you and you alone.

Homework

- Homework will be due at 3:30pm on the specified due date for both sections.
- Please submit your homework solutions to the TA electronically via Questrom Tools and include all necessary code and data files.
- All code should be self-explanatory and easily re-runnable by me or the TA.
- Late homework submissions will result in a loss of 10 points for each day past the due date.

Questions?

Questionnaire

- In order for me to better understand your background, please complete the initial questionnair that has been posted to Questrom Tools:
  - What areas of finance are of most interest to you and why? (e.g. big data, trading, portfolio management, risk managements, derivative pricing, exotic options)
  - What is the primary reason for your interest in this course?
  - What skills are you looking to gain from taking this course?
  - List all the programming languages that you've used and your level of comfort with each.
  - Where are you located this semester? Will you be attending class remotely or in-person? If attending remotely, which section will you be attending?

Quant Landscape

Quants may find themselves at any of the following institutions:

- Sell Side: Bank / Market Maker / Dealer
- Hedge Fund
- Asset Manager
- Fin. Tech Firm

Within these firms you could end up in many different seats:

- Desk Quant
- Quant Developer
- Quant Porfolio Manager
- Researcher

Regardless of where you end up, you will most likely spend a significant amount of your time coding, and working with code.

What types of problems do quants solve in these roles?

- Sell-side desk quant

  Pricing & Greeks for Exotics

- Buy-side desk quant

  Regresssion / Machine learning / Time Series

- Risk quant

  VaR Modeling / Model validation

- Asset Management quant

  Optimization / Portfolio Construction

- Research quant

  Alpha generation / Estimation / Filtering / Model design

What types of problems do quants solve in these roles?

- In general, quants are trying to compute the expectation of some quantity (e.g., an asset price)
- In order to do this, a quant will always rely on a simplified model of how the world works.
- The quant will need to identify how to calculate this expectation under the simplified model and will need to write a suitable piece of code to calculate the expectation.
  - Perhaps the most common example of this is the Black-Scholes Model

What types of problems do quants solve in these roles?

- In this course we focus less on how to specify and derive the model and more on its implementation.
- You should still be able to gauge the strengths and weaknesses of the differents models that we use and be able to decide which ones to use in different applications later in your career.
- **Question**: What are some strengths and weaknesses of the Black-Scholes model? How about the Bachelier model?

What do these quant problems have in common?

- You will find almost all quant problems will not have a closed form solution. As a result, coding an approximation will be of critical importance.
- At the center of this process is an underlying set of programming skills & concepts. In particular we need to be able to:
  - Gather, clean and store the appropriate data. This means we need to know how to interact with databases and languages that are able to parse data conveniently.
  - Write a piece of code that efficiently implements our model of choice. As the model becomes more complicated, you will see that this step becomes more complex.
  - Create a series of tests that prove that our model are accurate and robust. This is called **model validation**.

What does a typical quant project look like? First Example: Option Pricing

- Suppose you are asked to price an American digital option, which is an option that pays $1 if an asset price touches a certain barrier level *at any point* in its path.
- This is similar to a European Digital option which pays $1 if an asset price touches a barrier at expiry.
- In FX, these are referred to as **one-touch** and **no-touch** options.
  - There are also **double no-touch** options which pay if the exchange rate never leaves a given range.
  - European touch options only observe the barrier at expiry.
  - American touch options observe the barrier at every tick.
- Question: How would you go about this?

What does a typical quant project look like? First Example: Option Pricing

- American digital options are **path dependent**. Therefore, modeling them requires a model that fits the entire volatility surface.
- The first step would be to write a program that automatically **gathers the data** for all quoted options on the underlying asset.
- The next step would be to **store the data** in some sort of **database**.
- The next step would be to **clean the data**. This should include checks to make sure that the data is **arbitrage-free**.
- The next step would be to implement an options pricing model that can handle pricing American options. For example, we could use a simulation approach and the Black-Scholes model

Question: If we use this approach, would we be able to match the entire volatility surface?

- Then we would construct a set of tests to ensure that our code produces the correct price and is reliable.
- The final step is then to make the model results available to the desired user(s).

Lifecycle of a Quant Software Project

- Stages of a quant software project:
  - Data Parsing / Collection
  - Data Cleaning
  - Database Design & Data Storage
  - Model Implementation
  - Model Validation
  - Presentation of model results via an interface
- In this course we will provide the tools for completing each step in this process in a robust manner.
- No matter where you end up on the landscape of quants, it is likely that you will spend a significant portion of time working on each of these steps.

Overview of Coding Languages

The most common programming languages used in finance are:

- Python
- C++
- C#
- Java
- MatLab
- R

In this course we will focus exclusively on Python and C++.

Overview of Coding Languages

- Broadly speaking, programming languages can be broken down into:
  - Low Level Languages (i.e. C / Assembly)
  - High Level Languages (i.e. C++ / C# / Python / R / MatLab)
- High Level Languages must translate the source code that the programmer writes into machine code before the program can be run. This can be done
  - Interpreted: each line is translated as it is run
  - Compiled: the entire program is translated as it is compiled

Strengths and Weaknesses of Coding Languages

- C++ is a common language for high performance applications, such as high-frequency trading. It is a more difficult language to code in, but enables the programmer to write far more efficient code.
  - Banks, hedge funds and other firms also have a fair amount of legacy quant applications that are written in C++.
  - Because of this C++ still plays a key role in the quant finance industry
- Python, R and MatLab are all common languages for statistical analysis and lower frequency trading models.
- C# and Java are commonly used for interface design and for building data collection and data cleaning scripts.

Strengths and Weaknesses of Coding Languages

- C++, Python, C# and Java are the best for **object-oriented programming**.
- MatLab and R are functional programming languages, although they both have some support for classes.
- Python has become a very trendy language of late because of its ease of use in statistical applications, its powerful IDE's and its support for object oriented concepts.

What is object-oriented programming?

- Object-oriented programming is based on building a set of objects, or classes each of which may have their own **member data** and thier own **functions**.
- For example, we may define a *Position* class that defines certain common attributes, such as shares and market price.
- We might also define a function, *MarketValue* which computes the market value of every position every day. This function might vary depending on the type of underlying instrument.
- As we will see later in the course, using this type of framework enables us to write more generic code.

Environment Setup

- In this course you will be asked to program in both Python and C++.
- For Python, I highly recommend using the Spyder IDE, which is available for Windows and Mac OS and can be downloaded here: https://www.anaconda.com/download/
- For C++, I recommend you use Microsoft Visual Studio Community Edition, which can be downloaded here: https://visualstudio.microsoft.com/downloads/

Debugging

- Debugging enables you to step through your program line-by-line or function by function. It also enables you to run the code until an error occurs, or until a certain **breakpoint** is hit.
- Debugging is a critical component of any programming language.
- Writing large-scale applications without a seamless debugging experience is almost impossible.
- Python and C++ both have robust debugging capability. Keep this in mind when choosing a language (and a development environment for that language).

Modules in Python

- Python has many useful modules that you can install using the **import** command.

```
1 import numpy as np
2 import pandas as pd
```

- Numpy and pandas in particular are useful in financial applications and are worth reviewing.
- This is similar to R or MatLab which also have many additional add-on packages.

Working in a Multi-Programmer Environment

- When programmers work together, they need a means of coordinating their changes to the code and ensuring that the changes that they make don't break others work.
- Many professional programs are available to help with this. Perhaps the most common is git/github.
- We will only discuss the basics of git. It enables a high-level of customization if desired. More details can be found here for those interested: https://git-scm.com/doc
- I would highly recommend using git on your final project.

How does git help us?

- **Version Control**: Using git, or another source control solution provides us a way to keep track of the changes made to our codebase in an organized way.
- If we make a mistake in our coding process, git enables us to **revert** back to a previous working state.
- It gives coders a means for working together seamlessly. In particular it enables multiple coders to work on the same project, even the same file, and then **merge** their changes once they have completed their work.
- Production systems typically have a strict **release process**. git enables us to keep historical code for all previous production versions.

git Repositories

git has 3 layers in its version control system:

- Local Working Copy
- Local Repository
- Remote Repository

Main git commands

- **git status**: lists all changes you have made in your local files and local repository.
- **git add**: adds a new local file to your local repository.
- **git commit**: takes changes to all local files and commits them to your local repository (where they are still only visible to you).
- **git push**: pushes all committed changes from the local repository to the remote repository.
- **git pull**: retrieves changes from the remote repository.
- **git merge**: merges your changes with changes from a remote repository.

First Homework

- Your first assignment has been posted and is due September 24th.
- Remember to submit your code as part of your homework submission.
- Also remember that all results should be reproducible should the TA choose to run your code.
- Your homework will require knowledge of monte carlo simulations and options pricing, so let's spend a few minutes reviewing the underlying concepts to give you a head start...

Review: Black-Scholes & Bachelier Models

- **Black-Scholes Model (Log-Normal)**

$$dS_t = rS_t\,dt + \sigma S_t\,dW \tag{1}$$

  - Asset prices are log-normally distributed.
  - Model parameters are r and σ.

- **Bachelier Model (Normal)**

$$dS_t = rS_t\,dt + \sigma\,dW \tag{2}$$

  - Asset prices are normally distributed.
  - Model parameters are r and σ.
  - Model is most commonly used in interest rate markets. (Why?)

- Question: What is the difference in the σ's between the two models?

Black-Scholes Formula

- There exists an analytic solution to (1), known as the **Black-Scholes formula**, for the price of a European call:

$$c_0 = \tilde{\mathbb{E}}\left[e^{-rT}\left(S_T - K\right)_+\right] = \Phi(d_1)S_0 - \Phi(d_2)Ke^{-rT} \tag{3}$$

- Here, Φ is the CDF of the standard normal distribution and

$$\begin{aligned} d_1 &= \frac{1}{\sigma\sqrt{T}}\left(\ln\frac{S_0}{K} + \left(r + \frac{\sigma^2}{2}\right)T\right), \\ d_2 &= d_1 - \sigma\sqrt{T} \end{aligned} \tag{4}$$

- In theory, we can price European options with constant short rate and constant volatility analytically using the Black-Scholes formula

Black-Scholes formula: Implied Volatility

- In reality, we generally observe a market price and need to find the σ in equation 1 that causes the market price to be matched.
- To do this, we need to solve 3 for σ. There is no closed form solution to this and must be done numerically.
- We can write:

$$c_{0,K} = BS(K, \sigma) \tag{5}$$

  which tells us that σ is a function of K. It is called volatility (or vol, for short) of K.
- To compute vol of K from the price, we will need some root-finding method. One important example is Newton-Raphson.
- This process of fitting model parameters to market prices is called **calibration**.

What causes an option price to change in the Black-Scholes model?

- **Delta**: A move in the underlying asset, S_0
- **Vega**: A move in the implied volatility of the option, σ
- **Rho**: A change in the risk-free rate, r
- **Theta**: A change in time to expiry, τ
- This leads to a set of sensitivities, often referred to a set of Greeks that are pervasive throughout the field of quant finance. It is worth becoming familiar with them.

Comments on Black-Scholes Model Sensitivities

- One may notice that the set of sensitivities that we care about is **model dependent**.
- The relationship between an option price and the underlying asset is not linear, leading to a second-order greek, **Gamma**.
- You may also have noticed that the σ, or implied vol. that we calculated was a function of the strike, K. What does that tell us?
- Note that in the Black-Scholes model a change in the underlying asset is assumed to be independent of a change in its implied volatility. Does this seem reasonable to you?

Review: Simulation

- Simulation is the process of using randomly generated numbers in order to approximate an expectation.
- The idea behind simulation is that we **randomly sample** from some distribution **repeatedly** in order to get an estimate of some function of that distribution.
- There are some theoretical results that tells us that if this is done properly, the distribution of our sample will converge to the true distribution as we increase the number of draws.

Review: Simulating Gaussian Random Variables

- Simulation of any stochastic process that involves a Brownian Motion term will require randomly sampling from a Gaussian distribution.
- Knowledge of simulating from a Gaussian distributions enables us to simulate the increments of 1 and 2 respectively.
- This enables us to generate paths of the underlying asset, and compute various desired expectations.
- Most programming languages have built-in functions for generating normal random variables. In python, the **numpy package** has a function called **random.normal** which does this.
- An alternative approach is to start by generating uniform random variables and then transform them to random normals.

Good luck!

It is going to be a fun course