Chapter 1. Introduction

"Price is what you pay. Value is what you get." Ben Graham[1]
"The most important single aspect of software development is to be clear about what you are trying to build." Bjarne Stroustrup[2]

Learning objectives

- Understand several compelling reasons why aspiring financial analysts should learn a computer language
- Enumerate multiple arguments for selecting R as the language of choice for financial analysts
- Introduce the R language (see 1.0R Introduction)
- Defend the unique approach to learning R provided in this material

Introduction

The purpose of this book is *not* to provide state-of-the-art R programming techniques. The purpose of this book is also *not* to provide state-of-the-art quantitative finance techniques. Financial quantitative analysts (quants) often lack the foundational understanding of quantitative finance as well as basic R programming. Many quants have studied the graduate level quantitative finance textbooks and passed various quantitative finance-type examinations. They have not, however, seen how to deploy these ideas in practice. Thus, this material seeks to fill this void by providing a launching pad where quantitative finance professionals can connect the dots between abstract theoretical finance concepts and prototype code that could be used to implement various quantitative finance ideas.

The purpose here is to provide as simple approach as possible to enable financial analysts to develop prototype implementation R code of their quantitative finance ideas. The premise here is that two novices in R can solve difficulties in their work for one another better than the R master can. The objective of this book is to help with quantitative finance professionals who wish to implement their emerging quantitative finance ideas with R. Alternatively, we aid with the computer programmer or mathematician who wish to understand quantitative finance applications. We focus on introducing quantitative finance with R because many of the modern quantitative finance concepts require some form of computer program to successfully implement.

To ease the exposition, this material is broken into two separate sets of materials, quantitative finance and R commentaries. Thus, those proficient in R programming (or Python, C++ and so forth) can focus on the quantitative finance and those proficient in quantitative finance can focus on R programming.

Computer programming is a unique, disciplined implementation of ideas. In this chapter, several reasons why financial analysts should learn a computer language are reviewed. In particular, the case will be made for R and it will be briefly introduced.

---

[1]Warren Buffett in his 2008 letter to Berkshire Hathaway shareholders attributes this quote to Ben Graham.
[2]Quoted in Herb Sutter and Andrei Alexandrescu, *C++ Coding Standards 101 Rules, Guidelines, and Best Practices* (2005), p. 55.

All the R code illustrated in this book, as well as much more, is available at http://www.robertebrooks.org/project/buildiingqfawr/. We now discuss the social science nature of quantitative finance.

Quantitative finance and social sciences

Quantitative finance falls within the realm of social science, and not natural science. For example, to provide quantitative support for a large, and complex option trading firm, a clear understanding of the social science nature of this activity is vital. Natural science quantitative work can focus on accepted solutions: Once well tested in the lab, it will always hold. Social science quantitative work is distinctly different in that human behavior and human understanding can change rapidly, hence, the very structure of the quantitative problem changes. Security and derivative prices are inherently influenced by market participants' views. These views incorporate beliefs about future trends, beliefs about future opportunities and disasters, and even beliefs about potential future valuation models.

Because of the social science nature of finance, the quants focus is expected to be on developing an infrastructure that can adapt quickly to paradigm shifts in human understanding of pricing and human preferences for products. Hence, the quant should always place high value on scalability and interchangeability. A firm needs to be able to manage rapid expansions and contractions in business as well as be able to manage rapid changes in solution technologies that work.

For example, as option markets consist of people, option models need to reflect the present driving forces motivating the trading behavior of people. Because this activity falls more in the realm of social science than natural science, an option model's usefulness will depend upon a particular market and a particular season of time.

Thus, the value of a financial organization depends upon an infrastructure that results in a continuous flow of innovations to models as well as numerous other complex infrastructure issues.

The development of financial innovations typically requires the interplay between theoretical models and empirical data. It is difficult to develop useful models without access to high quality market data. The mere existence of high quality market data does not result in the development of useful models. To have a chance of developing useful models in financial markets, there must be the interplay of researchers with high quantitative skills, and practitioners with a deep understanding of markets. Both are usually required.

Why a financial quantitative analyst should learn a computer language

There are several reasons for a financial quantitative analyst to learn a computer language such as R. These reasons include being better able to avoid conformity to popular black box solutions, developing a precise understanding of posited models, improving the buy versus build decision-making process, providing better solutions than are possible with spreadsheets or symbolic languages, improving model debugging, and learning to decompose complex problems into manageable components. We explore each of these reasons in more detail.

Conformist versus non-conformist

There are many influences in the financial quantitative analyst profession that result in entering professionals conforming to industry standards. There is a unique professional language and standardized methods of expressing yourself (for example, client presentations, accepted valuation methodologies, expected historical statistics gathered, and standardized analysis tools, such as Bloomberg®). Every analyst, however, is unique and brings to the profession a distinct perspective that may prove very valuable to her long run success.

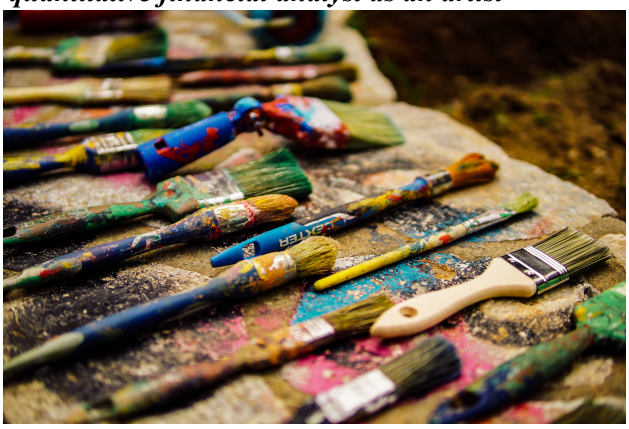
One way to preserve an analyst's unique perspective is by developing non-standardized valuation and management tools via a computer language. Knowledge of a computer language dramatically expands the analyst's means of expression. R allows the analyst to quickly develop innovative tools.

An important contribution of analysts providing independent perspectives is the reduced likelihood of systemic events. If every analyst is performing their tasks in the same way, then the likelihood of systemic events increases. By equipping analyst with the power to implement independent perspectives, the global financial system becomes more robust.

By analogy, the analyst is in some ways an artist. Learning to code in R is simply learning different basic paint brushes as well as brush strokes. As illustrated in Figure 1.1, we hope you eventually develop into a skilled quantitative finance artist able to rapidly build unique solutions for the numerous and rapidly

changing problems. We will explore a variety of unique canvases (different applications such as valuation, static risk management, and dynamic risk management), a variety of unique paints (different solution methodologies, such as those based on geometric Brownian motion and arithmetic Brownian motion[3]), and paintbrushes (different ways to code in R, kept very basic here).

Figure 1.1. Illustration of a quantitative financial analyst as an artist[4]



Clear and crisp understanding of model

For most applications related to quantitative finance, a computer program that is 99% correct is 100% wrong. That is, an error in implementation is likely to show itself at the very time the program is most needed to be correct. Many financial quantitative analysts do not have a clear and crisp understanding of the quantitative models and techniques they use. It is analogous to observing land contours from 30,000 feet in a plane. Although everything may look fine and smooth from a high altitude, descend to 500 feet and the land contours change dramatically. Programming leads to a more precise understanding of the ground contours of quantitative models and techniques and provides a very detailed level of understanding.

For example, one way to value an interest rate swap is as a portfolio of forward rate agreements. At the highest level, an interest rate swap is simply the present value of forward rates. Plain vanilla interest rate swaps, however, widely reported is a semi-annual, 30/360 day count fixed rate and a quarterly, Actual/360 day count floating rate. The differences in payment frequency and the day counting have a significant impact on equilibrium swap rates. Thus, by learning to write a computer program, the analyst will have a much better understanding of the intricate details of actual interest rate swaps.

Build versus buy

The decision to express quantitative finance ideas in a computer language has several advantages and disadvantages. The disadvantages include the time and energy that is required to complete the tasks. One alternative is to simply purchase computer software that provides quantitative solutions ready to be deployed. In the short run, this solution is often very attractive in that one simply pays an *immodest* fee and within a very short period, the quantitative finance models are up and running. In many cases, this is a reasonable solution.

Unfortunately, one side effect of buying software solutions is that no one internal to the entity fully understands the nuances of the deployment. Many decisions that are made when developing software are not expressed in public documentation. Hence, purchased quantitative finance software always has an element of

[3]Please do not panic if we illustrate a solution methodology that you or your instructor may deem heretical. You can choose not to use those paints. Often, however, the solution methodologies presented here are absent from academic writings, but have been found deeply useful by practicing quantitative finance professionals.

[4]This figure, as well as many others in this material, was obtained at www.unsplash.com (a copyright free warehouse of pictures). Photo by Yannis Papanastasopoulos on Unsplash.

being a black box. The best that the financial analyst can do using this purchased software is understand the required inputs and strive to accurately interpret the reported outputs.

On the other hand, the decision to build software internally has the advantage of potentially being fully understood by the financial analysts within the firm. The software can be modified as market conditions change, and improvements are made. Model maintenance becomes more of a process rather than a single static decision. Because finance is a social science[5], maintaining flexibility to modify model design is very useful.

Admittedly, the decision to build software internally has unique disadvantages, such as implementation errors. Thus, one significant advantage of purchasing, as opposed to constructing software, is that implementation problems are the responsibility of someone else.

Computer language or spreadsheets

Many financial firms rely heavily on spreadsheets for their required analytical work. Unfortunately, a common repercussion is that the only solutions posited for quantitative problems are those that can be implemented within spreadsheets. When seeking the best solution for difficult problems, one would rather have a more extensive set of solutions. The spreadsheet paradigm can limit one to the finite number of feasible solutions that are available to solve quantitative finance problems. Indeed, for large and complex problems, spreadsheet solutions can become very cumbersome and difficult to manage. With a computer language, one can easily decompose complex problems into component parts and build software solutions that are manageable. Many computer languages, such as R, are optimized for numerical calculations and as a result are fast.

Finally, modules can be developed in R and other computer languages that run within spreadsheets, thus providing the best of both worlds. These modules are typically user-defined functions facilitated by dynamic-linked libraries or some other linking facility.

Computer language or symbolic languages

Symbolic languages provide a useful solution to many mathematical problems. Unfortunately, modules developed through symbolic languages are not portable to computer hardware that does not have the symbolic language software installed. As R is free, portability is not a problem. Most computer language compilers, such as C++, produce solutions that are portable to any computer that contains the appropriate operating system.

In other words, if a quant writes a solution to a quantitative finance problem in a symbolic language and then wishes to provide it to her superiors, they would be unable to run the program on their machines unless the machine contains the symbolic language. Given that these symbolic language programs can be costly and take up a lot of memory, it can be cumbersome to have them housed on each machine within an entity.

Improved communication

Software developers within finance organizations often have no formal training in finance. Although they may be very efficient and effective in rapid application development, often they do not understand and fully appreciate the nuances of advanced quantitative finance applications. Hence, one valuable service provided by financial analysts with a familiarity with computer programming, is the ability to enhance the information flow between the finance professional and the software developer. This skill is extremely valuable as it further refines the subsequent flow of information from finance professional to senior management. As overall communication between these disparate groups of professionals improves, the likelihood that the project gets deployed on time with minimal problems improves as well.

More efficient debugging

Quantitative finance applications are complex for many reasons, including the need for speed, real time data management, and multidimensional and advanced mathematical solutions. The finance professional often does not fully appreciate the importance of validating the accuracy of newly-implemented software. Many errors in programming, referred to as bugs, do not appear until a crisis occurs. For example, during the

[5]This assertion stems from the following logical arguments: 1) Individuals' conception of the world has the capacity to affect real change in the marketplace (a phenomenon identified as performativity). 2) There naturally exists a degree of ebb and flow in societal value systems. 3) Thus, finance should be studied as a social science rather than a physical science.

financial crisis that started in 2007, many flaws in model implementations came to light. Unfortunately, fixing flaws is difficult when your firm is in crisis mode.

By investing in your capacity to understand computer program code like R, you will enhance the likelihood of identifying problems before it is too late. Understanding the computer programming process improves one's ability to know where errors are likely to occur.

On September 9, 1947, Harvard University operators of a crude calculator were experiencing technical problems. They found a moth at Relay #70, Panel F on the Mark II Aiken Relay Calculator. In their journal entry they indicated that they had "debugged" the machine. Hence, correcting problems with computer code is often referred to as debugging. The picture is a photograph of the actual moth recovered from the relay calculator. (See https://www.history.navy.mil/content/history/nhhc/our-collections/photography/numerical-list-of-images/nhhc-series/nh-series/NH-96000/NH-96566-KN.html.

Decomposition

The process of learning a computer programming language enhances the financial analyst's ability to decompose complex problems into manageable components. Often this process is referred to as decomposition. Decomposition is the exercise of breaking a problem down into smaller, manageable pieces. Computer languages, such as R, aid in developing skills to achieve the optimal level of decomposition. Often insurmountable problems become easily manageable when the problem is decomposed appropriately.

Where we go from here

In Part 1, we will explore three categories of approaches to valuation in Chapter 2 and review several important concepts and quantitative tools in Chapter 3. Starting in Chapter 3, we will take a modular approach with an effort to make each module stand alone. When a module requires a function covered previously, the R code will be copied to that module easing the ability to run each module program separately. In Part 2, we cover valuation starting with bonds and stocks in Chapter 4. In Chapter 5, we cover option valuation. In Chapter 6, we cover forwards, futures, and swap. In the next two book Parts, we cover the same instrument categories in four chapters each. In Part 3, we examine various static risk measures. Static risk measures can be thought of as mathematical derivatives such as delta, the first derivative of the instrument with respect to the underlying. In Part 4, we examine various dynamic risk measures. Dynamic risk measures typically involve changing several input variables simultaneously using tools such as Monte Carlo simulation. Value-at-risk is an example of a dynamic risk measure. We conclude the book in Part 5 by addressing portfolio issues in Chapter 14.

Summary

We began this chapter by exploring several compelling reasons why aspiring financial analysts should learn a computer language. These reasons include: to enhance one's ability to express unique analytical ideas, to improve one's understanding of financial models, to provide the opportunity to build rather than buy software, to enhance spreadsheet or symbolic language development, to improve communication with internal software developers, to provide more efficient debugging within a firm, and to enhance one's ability to decompose complex financial problems into manageable parts.

References

Chance, Don M. and Robert Brooks, *An Introduction to Derivatives and Risk Management*, 10th Edition (Mason, OH: South-Western Cengage Learning, 2016).

Sutter, Herb and Andrei Alexandrescu, *C++ Coding Standards 101 Rules, Guidelines, and Best Practices* (2005).