Chapter 1

Introduction

1

Book Overview

- Introduction (this presentation)
- R Preliminaries (in appendix)
- Quantitative finance tools
- Instrument valuation
- Static risk management
- Dynamic market risk management

2

Introduction to Introduction

- Purpose: Not to provide state-of-the-art R programming techniques (provide selected)
- Purpose: Not to provide state-of-the-art quantitative finance techniques (provide selected)
- Purpose: Provide as simple an approach as possible to learn prototype implementation code
- Facilitate implementation of quantitative finance ideas in R

3

Case for Computer Language

- Conformist versus non-conformist
- Clear and crisp understanding of model
- Build versus buy
- Computer language (CL) or spreadsheets
- CL or symbolic languages
- Improved communication with internal software developers
- More efficient debugging
- Decomposition

4

Conformist vs. Non-Conformist

- Unique language
- Methods of expression
  - Client presentations
  - Valuation methodologies
  - Data collected
- R expands means of expression
- Improved software tools speeds the process

5

Clear / Crisp Understanding

- Computer program – 99% correct is 100% wrong
- 30,000 foot perspective of financial models
- Plain vanilla interest rate swap
  - Quarterly, ACT/360 FLT; Semi, 30/360 FIX
  - Payment frequency and day count have significant value

6

Build Versus Buy

- Build
  - Takes time and energy, risk errors but …
  - Fully understood by someone within firm
  - Easy to modify as conditions change (social science)
- Buy
  - Pay immodest fee, someone else liable, fast but …
  - No one internal understands model nuances
  - Black box (only know inputs and interpretation of outputs)

7

Spreadsheets

- Analysis only compliant with spreadsheet framework (all other solutions not considered)
- Large, complex problems difficult in spreadsheets
- Slow and cumbersome
- CL: Decompose large, complex problems
- CL: Fast and flexible

8

Symbolic Languages

- Not portable
- Rely on existence of symbolic language provider
- CL: very portable
- CL: rely only on existence of language, not specific compiler

9

Improved Communication

- Internal software developers do not understand finance language
- Advanced quantitative finance applications often are fraught with nuances (social science)
- Financial analyst can better communicate with internal software developers if understand computer language

10

More efficient debugging

- Quantitative finance models are complex (speed, real time data, multidimensional, advanced math)
- "Bugs" are rampant 
- Financial analyst is well-suited to efficiently debug

11

Decomposition

- Breaking down problem into smaller manageable pieces
- One goal here is to help you develop skills to achieve optimal level of decomposition

12

Why Learn the R Language?

- Jobs
- History of computer programming languages
  - Low level language: machine level code, powerful
  - High level language: easy to use
  - mid-1950: FORTRAN (formula translation), high level language
  - C++: combines high level (easy to use) and low level (powerful), fast, object-oriented
  - R: Easy and can link with C++
- Rapid application development

13

Learning R

- Autonomous versus heteronomous
  - Autonomous: Freedom to act independently, training materials are compiler independent
  - Heteronomous: Subject to external standard, training materials specific to one compiler
- Deliverables: Simple prototype programs
  - Actual implementation requires exhaustive error-trapping, real time data
  - Goal: Deployable code

14

Summary

- Case for learning computer language
- Why R rather than spreadsheets or symbolic languages
- Learning R improves communication, more efficient debugging, decomposition, and jobs

15

Appendix: Building a Repository

- Managing subdirectories for source code
  - C:\QFRepository
- Managing files
  - Modular development
  - Each module independent

16

Appendix: Coding Preferences

1) Indent two spaces (even for wrapped lines) after each curly bracket ({), method and class curly brackets place on separate line, all others open ({) on same line and close (}) on new line
2) Use blank lines very rarely. If you need a space, include a comment line
3) Your name should be on the first line of each file

17

Coding Preferences Continued

4) R function code should be separated from other code
5) Naming conventions adopted here
   a. Names of variables: lower case or both upper and lower case, err on longer name
   b. Names of functions: begin with upper case for each word

18