`tradesys`: A framework for modelling trading systems in R

Robert Sams

April 13, 2012

1 Introduction

> Define your project and the right tool appears. Questions about the tools indicate uncertainty about the project.
> Ed Seykoda

The `tradesys` package is for modelling trading systems in R. The key functionality of the package is centered around the 'tsys' class. A 'tsys' object collects all of the information needed to completely define a trading system. The key ideas behind the system, like the entry and exit signals, are written by the user in R and stored as unevaluated R expressions in the appropriate 'tsys' slot. The object can subsequently be applied to any timeseries data with the appropriate variables, the logic of the system encoded in the stored expressions are evaluated on this data, and the system's long/short/flat *states* and the resulting *equity curve* are calculated. In short, a trading system consists of certain R-encoded logic and meta-data, defined in an object of class 'tsys', and evaluated on any data of the appropriate structure to calculate the system's states and equity.

So, the functionality of the package is modest in its scope. It is, however, ambitious in the implementation. The above model is needed in almost all trading system research and thus represents a problem in need of a common, well-designed solution. This package aims to do this to the highest standard, so that trading system builders who chose R as an important tool of analysis can confidently use this package a key component of their work.

There are three main design goals of the package, one bold, two conservative. First, the bold. The package aims at *maximum expressibility*. There is no one model that captures everything that is properly thought of as a trading system, but many (most?) trading systems share a common model and the "tsys" class aims to represent this in R. This goal is cast in the concept of "expressible" rather that "feature-full" because the class is designed with the notion that seemingly special aspects of the system that you want to model can be expressed in terms of simplier, basic ideas. For example, there is no explicit modelling of slippage assumptions in "tsys", yet the mechanism for putting data variables into a pricing

context implicitly allows the user to express just about any slippage model he wants. We'll show some slightly more exotic examples later in this document demonstrating how ideas like slippage, financing costs, dynamic hedging, etc can be modelled in the class by building on its simple but flexible components. (Users are encouraged to construct examples that challenge this framework so that this approach can be extended an improved in later versions.)

Second, the package should be *trustworthy*, in the sense elaborated by Chambers .... This is a conservative goal but arguably more important than the one just mentioned...

Third, *discoverability*. The logic behind every calculation should not only be scrupulously documented, but also discoverable, in the sense that the key computations done on 'tsys' objects are encapsulated in functions that the user can call and explore. So the package contains a number of functions that are not strictly speaking part of the user interface (the usage of the package can be had without ever calling them), but are documented seperately and exported by the package's namespace to be explored at will.

1.1 What about "backtesting"?

The `tradesys` package is *not* a "framework for backtesting". One often hears requests for such a thing among R users in finance. But there is nothing meaningful about a test on back data without a conjecture about some market anomoly and some ideas about a system that can exploit it with an edge. The appropriate tests are as varied as the conjectures, and so with the tools needed for the testing.

As it so happens, the tools needed backtesting properly conceived are either computationally easy well-supported in R (see the various packages for timeseries analysis, robust methods and, of course, the base language); it's formulating the conjectures and key tests that are hard. Hence, the Ed Seykoda comment opening this document.

Modelling the trading system, on the other hand, is conceptually easy but computationally tricky. Coding from scratch the relationship among a system's signals, states, equity curve, etc. is not only time-consuming but pregnant with risks of subtle, logical errors. Making this part easy and trustworthy is the motivation for `tradesys`.

But if what you are after is a mechanism for translating a system into your favourite suite of "performance measures", this package does 99% of what you need: just call your favourite functions on a 'tsys' object's equity curve. But please don't look here for the home of R's latest implementation of the Sharpe Ratio.

2 A formal definition of "trading system"

A *trading system* is an algorithm on a timeseries X_t that specifies, for each time t, whether the system's state is long, short or flat. Mathematically, it is a function

$f(X_t)$ that calculates each state $s_i \in \{1, 0, -1\}$ on the basis of $X_1, ..., X_i$. X_t may be as simple as a daily series of closing prices but is often a multivariate series with various price and other data. The states vector combined with the timeseries is the raw material for backtesting research from the calculation of period returns onwards. Let's call such a combination $\{X_t, s_t\}$ a *trading system time series*. In this package a trading system time series is represented as class `tsts`.

But what about $f(X_t)$, what form does it take?...

3 Introductory Examples

4 Splicing Timeseries

Computational details

The results in this paper were obtained using R 2.15.0 with the packages `tradesys` 0.1 and `zoo` 1.7–7 R itself and all packages used are available from CRAN at `http://CRAN.R-project.org/`.