`tradesys`: A framework for encoding and backtesting trading systems in R

Robert Sams

July 17, 2009

1 Introduction

Design goals: maximum expressibility and tight integration with core R.

Mention other related packages like blotter, TTR, xts, etc...

1.1 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 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 X_t^v with various price and other data. The states vector combined with the timeseries is sufficient to do most analyses, like period returns, drawdowns, etc. Let's call such a combination $\{X_t^v, s_t\}$ a *trading system time series.* It can be thought of as the graph of a trading system function when applied to a specific timeseries X_t In this package a trading system time series is represented as class `tsts`.

But the trading system itself, the function from data to states, is encoded using `tradesys`. This is best explained by working through a simple and concrete example.

1.2 Example: Dual-moving average system

```
> library(tradesys)
> library(TTR)
```

This system is long whenever the 60-day moving average of price is above the 120-day moving average and short otherwise. We test it on the S&P 500 index.

```
> data(spx)
> colnames(spx)
```

```
[1] "Open"   "High"   "Low"    "Close"  "Volume"
```

The sample dataset spx is a zoo matrix and contains daily OHLC and open interest data for about 60 years. The system can be defined in one simple call to tradesys.

```
> x <- tradesys(data = spx, el = SMA(Close, 60) >= SMA(Close, 120),
+     es = SMA(Close, 60) < SMA(Close, 120))
```

The el and es parameters define the system's long and short entry criteria, respectively. They take expression objects that must evaluate to logical vectors equal in length to nrow(data). The expressions are evaluated in the normal way using R's *lazy evaluation* scheme, although tradesys first puts the columns of data into the evaluation frame as named vectors, so Close in the above expression evaluates as if it were data[, 'Close'].

So what did it return?

```
> class(x)

[1] "tradesys" "tsts"

> tail(x)

             Open   High    Low  Close     Volume states
2009-05-12 910.52 915.57 896.46 908.35 6871750400     -1
2009-05-13 905.40 905.40 882.80 883.92 7091820000     -1
2009-05-14 884.24 898.36 882.52 893.07 6134870000     -1
2009-05-15 892.76 896.97 878.94 882.88 5439720000     -1
2009-05-18 886.07 910.00 886.07 909.71 5702150000     -1
2009-05-19 909.67 916.39 905.22 908.13 6616270000     -1
```

'tsts' class. We won't go into the details of these two classes just yet. Suffice it to say that x is data with the state vector calculated from the entry critieria cbinded to the right.

The analysis function equity is used to calculate period returns and the equity curve.

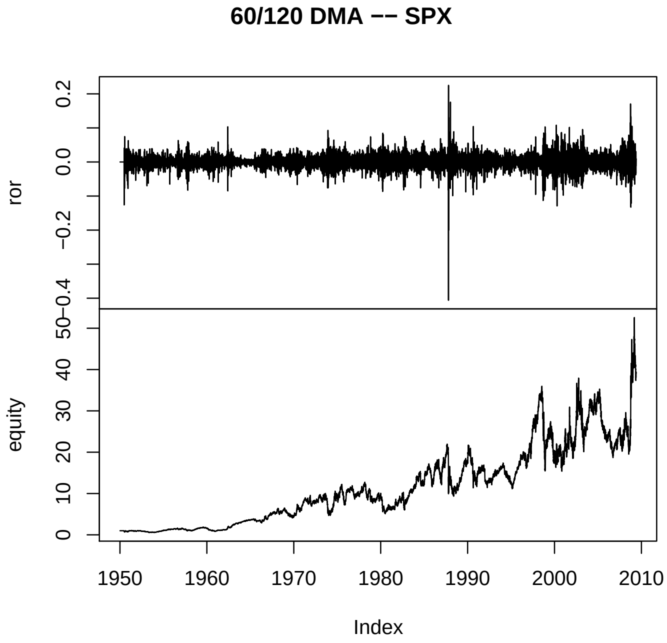
```
> y <- equity(x, uselog = TRUE)
> tail(y)

           trade states    delta    price          ror   equity
2009-05-12   112     -1 1.331220 6.814016  0.018107970 38.02601
2009-05-13   112     -1 1.307543 6.808377  0.007373275 38.30638
2009-05-14   112     -1 1.297973 6.784729  0.030694874 39.48219
2009-05-15   112     -1 1.259318 6.794318 -0.012075942 39.00541
2009-05-18   112     -1 1.274712 6.786796  0.009588169 39.37940
2009-05-19   112     -1 1.262606 6.813082 -0.033188777 38.07244

> EquityStats(y[, c("equity")])

    RORC     CAGR     ROR%       R2     VOLA    MAXDD
37.07244  0.06317  0.06213  0.84731  0.14157 -0.59800
```

```
> plot(y[, c("ror", "equity")], main = "60/120 DMA -- SPX")
```



Use `trades` to enumerate system trades and their holding period returns.

```
> z <- trades(x, uselog = TRUE)
> tail(z)

    phase      etime      xtime time nobs  eprice  xprice    pnl          ror
107    EL 2006-09-21 2007-09-12  356  244 1324.89 1471.10 146.21  0.264117052
108    ES 2007-09-12 2007-11-09   58   42 1471.10 1467.59   3.51  0.006027153
109    EL 2007-11-09 2008-01-03   55   36 1467.59 1447.55 -20.04 -0.034689959
110    ES 2008-01-03 2008-06-10  159  109 1447.55 1358.98  88.57  0.159301490
111    EL 2008-06-10 2008-07-21   41   28 1358.98 1261.82 -97.16 -0.187159273
112    ES 2008-07-21 2009-05-19  302  209 1261.82  909.67 352.15  0.825619186
```

What prices are assumed in these calcuations, as `spx` contains columns for open, high, low and close prices? By default, the prices used in these performance calculations is the left-most column, which is this case, is the open price. This won't due. We can't calculate our signal on Monday's closing price whilst trading on that signal using Monday's open price!

The `'tsts'` class has a very flexible mechanism for defining the price context of a trading system timeseries. We can set this from `tradesys` using the `pricecols` parameter.

```
> x <- tradesys(data = spx, el = SMA(Close, 60) >= SMA(Close, 120),
+     es = SMA(Close, 60) < SMA(Close, 120), pricecols = "Close")
```

This specifies that the system will assume that all trades are executed at the closing price. This is an improvement. But let's say that we want the system to compute signals on closing prices, position valuations at closing prices, and trades on the *following day's* open price. This (and much else) can be accomplished with the `makecols` parameter.

```
> x <- tradesys(data = spx, el = SMA(Close, 60) >= SMA(Close, 120),
+     es = SMA(Close, 60) < SMA(Close, 120), pricecols = list(long = "Next",
+         short = "Next", valuation = "Close"), makecols = list(Next = c(embed(Open,
+         2)[, 1], NA)))
> tail(x)
```

```
             Open   High    Low  Close     Volume   Next states
2009-05-12 910.52 915.57 896.46 908.35 6871750400 905.40     -1
2009-05-13 905.40 905.40 882.80 883.92 7091820000 884.24     -1
2009-05-14 884.24 898.36 882.52 893.07 6134870000 892.76     -1
2009-05-15 892.76 896.97 878.94 882.88 5439720000 886.07     -1
2009-05-18 886.07 910.00 886.07 909.71 5702150000 909.67     -1
2009-05-19 909.67 916.39 905.22 908.13 6616270000     NA     -1
```

`makecols` takes a list of expressions. These expressions are evaluated in the same manner as the `el`, etc., and their results are cbinded to `data`. Look at trade 111.

```
> trades(x)[111, ]
```

```
    phase      etime      xtime time nobs  eprice  xprice     pnl ror
111    EL 2008-06-10 2008-07-21   41   28 1357.09 1257.08 -100.01  NA
```

```
> window(x, start = "2008-06-10", end = "2008-06-11")
```

```
              Open    High     Low   Close     Volume    Next states
2008-06-10 1358.98 1366.84 1351.56 1358.44 4635070000 1357.09      1
2008-06-11 1357.09 1357.09 1335.47 1335.49 4779980000 1335.78      1
```

```
> window(x, start = "2008-07-21", end = "2008-07-22")
```

```
              Open    High     Low Close     Volume    Next states
2008-07-21 1261.82 1267.74 1255.70  1260 4630640000 1257.08     -1
2008-07-22 1257.08 1277.42 1248.83  1277 6180230000 1278.87     -1
```

Here you can see that the long entry on 10 June was done at 11 June's open price, and the long exit on 21 July was done at the 22 July open price.

The list passed to `makecols` is evaluated *before* the entry and exit expressions. They can therefore be referred to in the entry and exit expressions....

Computational details

The results in this paper were obtained using R 2.8.0 with the packages tradesys 0.1 and zoo 1.5–4 R itself and all packages used are available from CRAN at http://CRAN.R-project.org/.