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

Robert Sams

July 22, 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 $f(X_t)$ 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?...

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 dataset `spx`, which is a zoo matrix and contains daily OHLC and open interest data for about 60 years.

```
> data(spx)
> tail(spx)
```

```
             Open   High    Low  Close     Volume
2009-05-12 910.52 915.57 896.46 908.35 6871750400
2009-05-13 905.40 905.40 882.80 883.92 7091820000
```

```
2009-05-14 884.24 898.36 882.52 893.07 6134870000
2009-05-15 892.76 896.97 878.94 882.88 5439720000
2009-05-18 886.07 910.00 886.07 909.71 5702150000
2009-05-19 909.67 916.39 905.22 908.13 6616270000
```

The system can be defined in three lines.

```
> x <- tsts(spx)
> addsig(x, "el", SMA(Close, 60) >= SMA(Close, 120))
> addsig(x, "es", SMA(Close, 60) < SMA(Close, 120))
```

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

The states vector is now defined and we can do some backtesting.

```
> tail(x)
```

```
             Open   High    Low  Close     Volume St
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
```

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

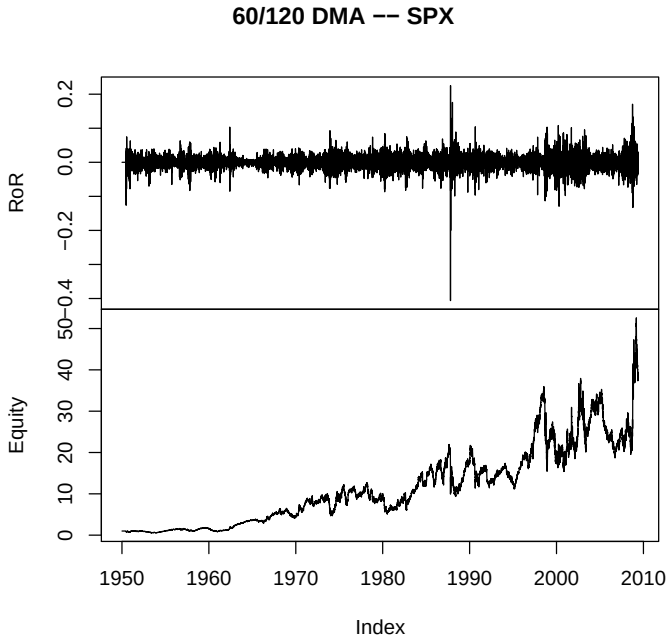
```
> y <- equity(x, uselog = TRUE)
> tail(y)
```

```
           Trade St    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 Numb  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.

```
> unlist(pricecols(x))
```

```
 valuation  enterlong entershort   exitlong  exitshort   rolllong  rollshort
         1          1          1          1          1          1          1
```

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.

```
> pricecols(x) <- "Close"
> unlist(pricecols(x))
```

```
 valuation  enterlong entershort   exitlong  exitshort   rolllong  rollshort
   "Close"    "Close"    "Close"    "Close"    "Close"    "Close"    "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 is a good place to introduce to the `addcol` function, for it is our solution to this problem.

```
> addcol(x, "Next", c(embed(Open, 2)[, 1], NA))
> tail(x)
```

```
             Open   High    Low  Close     Volume   Next St
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
```

So now `x` has an additional column named "Next", which contains the open price of the *next* day. Like the signal expressions, the expression that defines this column is stored in `x` so that the column is recalculated everytime new data is assigned to `x` via `coredata`. Now we can update our pricing assumptions with another call to `pricecols`.

```
> pricecols(x) <- list(valuation = "Close", entry = "Next", exit = "Next")
```

Let's Re-run `trades` and examine an arbitrary example, say trade 109.

```
> trades(x)[109, ]
```

```
    Phase      ETime      XTime Time Numb  EPrice  XPrice    PnL RoR
109    EL 2007-11-09 2008-01-03   55   36 1467.59 1444.01 -23.58  NA
```

You can confirm for yourself that the trade entry on 11-09 is done at the Open price on 11-12...

```
> window(spx, start = "2007-11-09", end = "2007-11-12")
```

```
             Open    High     Low   Close     Volume
2007-11-09 1467.59 1474.09 1448.51 1453.70 4587050000
2007-11-12 1453.66 1464.94 1438.53 1439.18 4192520000
```

```
> window(spx, start = "2008-01-03", end = "2008-01-04")
```

```
             Open    High     Low   Close     Volume
2008-01-03 1447.55 1456.80 1443.73 1447.16 3429500000
2008-01-04 1444.01 1444.01 1411.19 1411.63 4166000000
```

2 Splicing Timeseries

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/`.