

Performing equity investment simulations with the `portfolioSim` package

Kyle Campbell, Jeff Enos, David Kane, and Daniel Suo

September 18, 2007

Abstract

The `portfolioSim` package provides a flexible system for back-testing the performance of investment strategies using historical data. The package is designed to take into account the limitations of day-to-day trading, so that a simulation's performance will resemble as closely as possible the performance that a given strategy would actually have achieved over a given period of time.

This vignette documents several new wrapper functions that perform common procedures while removing the user from details concerning the R language. Examples will also detail new summary options that allow a by-period or by-security view of results.

1 Introduction

Note: This document has some overlap with the `portfolioSim` vignette, but is intended as a primer that describes the overall features of the package without delving into more advanced features and code. For more reading, please refer to the `portfolioSim` vignette.

Whatever investment strategy one uses to manage a portfolio, perhaps the most basic and most important question one can ask of it is: “How well does my strategy work?” Whenever one uses a systematic model to make financial decisions, one would like to have some assurance that the model will yield positive returns on an investment. Unfortunately, due to the unpredictability of the market, no estimates of a model's performance will ever be absolutely certain. The only sure indicator of an investment's value is the returns one can measure after the fact.

Therefore, barring the ability to foresee the future course of the market, the best way to gauge a model's accuracy is to examine its track record. Using historical data, it is possible to calculate how much money one would have made or lost had one invested according to a given strategy over some past period of time. This technique, known as “backtesting,” is widely used by financial professionals to test investment strategies. By making the assumption that there is at least some correlation between a model's past and future performance, investors can use the results of a backtest to make an informed decision on whether to use a strategy in their future investments.

The `portfolioSim` package is intended to give investors the tools to make such a decision. It does so by simulating a changing portfolio of investments over some period in the past. The simulator makes use of the `portfolio` package¹ to manage the portfolio. The simulator takes historical market data supplied by the user and follows an investment strategy, also provided by the user, to determine which stocks to trade and when to trade them. At the end of the simulation, the user is provided with detailed information on the portfolio's performance over that period. From this information, the user can determine whether or not the investment strategy was effective.

The `backtest` package² can be used to conduct one specific type of simple backtest for a single type of portfolio, specifically, a long-short portfolio formed from the highest and lowest ranked stocks of an input

¹See Enos and Kane, Analysing equity portfolios in R, for an introduction to the `portfolio` package.

²See Campbell, Enos, Gerlanc, and Kane, Conducting Backtests in R, for more information on the `backtest` package.

signal. The `portfolioSim` package, in contrast, is much more flexible, allowing the user to specify virtually any criteria for constructing and maintaining a portfolio.

In addition to being far more flexible in both the input it can receive and the output it returns, the `portfolioSim` package has several major advantages over the `backtest` package. One of the most important is its ability to step through time, period by period, making investments based on many market variables as they stood at some point in the past. The `backtest` package reduces the problem of forming a portfolio to a single ranking variable, buying the highest ranked stocks and shorting the lowest. But in reality, there are often many other factors that an investor must take into account. It may take several days to trade to a desired portfolio, and by the time all the selected stocks are actually acquired the prices may have changed and the stocks may no longer be desirable. The cost of making trades must be taken into account as well, along with the turnover in the portfolio.

The `backtest` package ignores all of these complications, and thus the results it reports, while useful for gauging the accuracy of a stock ranking, do not accurately reflect the returns one could expect to gain from actual investments. The `portfolioSim` package, on the other hand, is so named because it tries to simulate, as closely as possible, the day-to-day trading process as it occurs in reality. It allows the user to take into account daily trading volume, stock prices, trade-cost adjustment, portfolio equity, and many other such considerations that the `backtest` package ignores.

2 Running a simulation

At its core, the `portfolioSim` package is a very simple machine for maintaining a representation of a changing portfolio over some span of time. At any moment, this portfolio consists of holdings such as:

```
id shares
1 A      10
2 B      20
3 C     -10
```

Changes to a portfolio can be made in the form of “trades”.

trade: A transaction which changes the current holdings of the portfolio by means of buying, selling, shorting, or covering stocks.

Consider the follow list of trades:

```
id side shares
1 B   S      20
2 C   C       5
3 D   B      10
4 E   X      20
```

Applying these four trades to the holdings above would transform the portfolio into a new set of holdings, shown below:

```
id shares
1 A      10
2 C      -5
3 D      10
4 E     -20
```

Essentially, running a simulation consists of exposing a portfolio to different sets of trades at different points in time and observing the results. The duration of time over which the simulation is conducted is divided up by the user into discrete “periods”, at each of which a new set of trades is performed. The manner in which trades are selected for a given period is how we define an investment strategy. The user has three options:

1. Write an entire trading strategy as a trades interface from scratch that the package then queries for each period's set of trades.
2. Use one of several included trades interfaces to implement a pre-determined trading strategy.³
3. Use one of several convenience functions that assumes a trading strategy and creates an object ready for simulation.

3 Running a simulation

In this article, we will focus on the convenience functions to show the power of the `portfolioSim` package without getting bogged down in technical detail. Each of these functions takes a data frame, some requiring additional parameters, and creates a set of trades to perform in each period based on the given strategy. The resulting object can then be directly passed into the simulation function for processing. However, while different functions will require different data columns in the data frame, there are certain columns required by all tests. Thus, we begin with a general overview of the data frame preparation.

3.1 Preparing Data

The data included in the `portfolioSim` package was given by StarMine, a San Francisco based research company that creates rankings of stocks based on predicted future earnings. One such ranking is the StarMine Indicator, which ranks stocks on a scale of 1 to 100, with 100 being the highest.⁴ We can use the `stiFromSignal` interface to test the accuracy of the StarMine Indicator. If there is indeed a positive correlation between StarMine rankings and returns, we should expect a long-short portfolio formed by buying the highest ranked stocks and shorting the lowest ranked stocks to yield high returns. How well our portfolio performs should give us some idea of whether we wish to use the StarMine Indicator to make investment decisions.

The data set `starmine.sim` included in the `portfolioSim` package contains StarMine Indicator rankings for stocks from January 31, 1995 to November 30, 1995. In this data set, the rankings are updated monthly. All the other data we need to run a simulation are also included in the data set.

```
> data(starmine.sim)
> names(starmine.sim)

[1] "id"           "date"         "name"
[4] "country"     "sector"      "cap.usd"
[7] "size"        "smi"         "fwd.ret.1m"
[10] "fwd.ret.6m"  "price.usd"   "prior.close.usd"
[13] "volume"     "ret.1m"     "equal.weight.shares"
```

columns, but all require the following:

`period`: Most importantly, we need a column that identifies the period of each row. The column must be orderable, and in our case `period` is measured in months.

```
> starmine.sim$period <- starmine.sim$date
> sort(unique(starmine.sim$period))

[1] "1995-01-31" "1995-02-28" "1995-03-31" "1995-04-30" "1995-05-31"
[6] "1995-06-30" "1995-07-31" "1995-08-31" "1995-09-30" "1995-10-31"
[11] "1995-11-30"
```

³Please see Campbell, Enos, and Kane, Performing equity investment simulations with the `portfolioSim` package for more information on implementing ready-made trades interfaces.

⁴See www.starmine.com

id: Each row must also be identified by a stock `id`. The `period` and `id` columns combine to uniquely identify the data for each stock in each period. In our example, `starmine.sim` already includes an `id` column, so we don't need to change anything.

ret: We now need to select a column for the returns the simulator will use to calculate the portfolio's performance. The simulator uses the portfolio at the start of each period, before any trades are made, to calculate the performance for that period. In other words, we assume that it takes the entire span of a period for us to trade to a new portfolio. So even though we are trading on January 31, 1995, the simulator assumes that we do not get the portfolio resulting from those trades until the end of the period, on February 28, 1995. This is somewhat counter-intuitive, because we want to use returns from the month of February to calculate the performance of the portfolio that we traded to at the end of January (remember that we actually do all our trading on January 31, and then hold that portfolio throughout the rest of the period). To correct this, we use one month backward returns so that the performance for the period beginning on February 28 is calculated using the portfolio we formed on January 31 and the returns from the month of February.

```
> starmine.sim$ret <- starmine.sim$ret.1m
```

start.price: It is important to note that while the periods are spaced at monthly intervals, most of the columns in the data set refer to single days, not months. We must therefore consider carefully how we would go about trading in the real world, if we had only the information contained in this data set. Essentially, we are only allowed to trade on one day of every period, specifically the last trading day of the month. We therefore take the closing price of the day before the last trading day of the month as the `start.price`, and the closing price for the last trading day of the month as the `end.price`.

```
> starmine.sim$start.price <- starmine.sim$prior.close.usd
```

`portfolioSim` requires an `end.price` column as well.

```
> starmine.sim$end.price <- starmine.sim$price.usd
```

volume: In reality, trading can be limited by volume restrictions. `portfolioSim` handles this by filling an order to the maximum in a single period and saves the remaining order to fill in subsequent periods. In order to assess the limitations on trading, a `volume` column is required. In our example, this column is already included.

universe: Finally, we need to add a `universe` column to the data set. The `universe` column is a logical indicating which of the stocks the simulator will be allowed to trade. For the purposes of this example, we will assume that all the stocks for which we have data exist in the investable universe for that period.

```
> starmine.sim$universe <- TRUE
```

Now with our data correctly formatted, we can proceed with two examples.

3.2 An example with given rankings

One common method of testing a trading strategy is to assign some score or ranking to each stock in each period and then divide up all the stocks into some number of quantiles based on that ranking. If the stocks with a "good" score perform historically much better on average than stocks with a "bad" score, then most likely this trading strategy is good. Now let us apply this with a convenience function. Suppose we have some kind of ranking system for each stock in each period. For our example, we use the StarMine Indicator to rank stocks and divide them up into portfolios.

```
> signal.ps <- signal.given.template(data = starmine.sim, in.var = "smi")
```

We can also call the `summary` function on `signal.ps`. Again, we simply call the `runSim` method to run the simulation.

```
> signal.result <- runSim(signal.ps, verbose = FALSE)
```

It must be noted that this and any convenience function removes some flexibility from the user. For example, the `signal.given.template` function operates on one period's worth of data at a time, and assumes that data is collected and processed at a frequency of once per year. The function also saves all data by default. The resulting `portfolioSim` object we have called `signal.ps` can be manipulated, as we will show in the following sections, so most flexibility is still preserved, but more involvement is required. This is an unavoidable tradeoff between convenience and flexibility.⁵ Before continuing on to result reports however, we will go through another convenience function because they share the same summary options and then discuss how the user can manipulate the `portfolioSim` object returned by the convenience functions to better suit his or her simulation requirements.

3.3 An example with given holdings

Now suppose we know how many shares we own in each stock in each period and simply want detailed reports. To illustrate that both convenience functions are consistent with one another, we will use a column in the `starmine.sim` data frame called `equal.weight.shares` created by taking the share holdings in each period from `signal.given.template`.

The user also has the option to lag the trades by a number of months, positive or negative, to test the robustness of a sequence of trades. For this example, we will use the default of 0 lag.

```
> shares.ps <- shares.given.template(data = starmine.sim, share.var = "equal.weight.shares",
+   lag = 0)
> summary(shares.ps)
```

```
Data Interface:          sdiDf
NULL
Trade Interface:        stiPresetTrades
Lag:                    0
NULL
portfolioSim Summary:

Period Summary:
  First Period:          1995-01-31
  Last Period:           1995-11-30
  # Of Periods:          11
  Periods/Year:          1

Initial Summaries:
-----
  Initial Holdings:       4593
  Type:                  equal
Quantiles:               quintile
Sides:                   longSides:          short

Interface Summaries:
```

No summary interface specified.

⁵Please see Campbell, Enos, and Kane, Performing equity investment simulations with the `portfolioSim` package for more information on controlling all aspects of the simulation.

Other:

```
-----  
Exposure Variables:          None  
Contribution Variables:      None  
Fill Volume Pct.:           Inf  
Save Location:               sim_data  
Save Type:                   all
```

Now, to create our result object, we simply call the `runSim` method.

```
> shares.result <- runSim(shares.ps, verbose = FALSE)
```

3.4 Manipulating the portfolioSim object

Let us now first discuss the `portfolioSim` objects `signal.ps` and `shares.ps`. Both are called objects of class `portfolioSim` in much the same way another object might be called of class `integer` or `factor`. The `portfolioSim` object has many “slots” or private variables that can be set to change the result of the simulation that is run. First, let’s use the `signal.ps` `portfolioSim` object we made earlier. To access and manipulate a slot in an object, we use the `@` operator as shown below.

```
> signal.ps@freq
```

```
[1] 1
```

```
> signal.ps@freq <- 12
```

```
> signal.ps@freq
```

```
[1] 12
```

The `periods` slot is a data frame containing three columns: `period`, `start`, and `end`, defining the periods to be used and the start and end time of each. `freq` indicates the frequency of the periods per year; monthly analysis would mean that the frequency is twelve. `data.interface`, `trades.interface`, and `summary.interface` are the interfaces that retrieve data, return trades, and create the summaries, respectively. `start.holdings` indicates the initial portfolio. `exp.var` takes in a list of exposure variables and `contrib.var` takes in a list of contribution variables. `out.loc` indicates the location that the data will be saved, and `out.type` indicates the amount and type of data that will be saved. In the two convenience functions, these are set to `all` so we can take full advantage of the summary functions.

3.5 Summary Options

To illustrate the summary features, we’ll use the results from `signal.given.template`. The procedure is identical for `shares.given.template` and should be examined by the user to verify consistency. The overall summary includes data from the entire simulation and calculates many per period averages. First suppose, we would like to see a general summary:

```
> summary(signal.result)
```

Simulation summary:

```
Start: 1995-01-31
```

```
End:   1995-11-30
```

```
-----  
Total:                profit    return  
-----  
Total:                10,619,636    20.7 %
```

Sharpe:		1.2	

Mean return:		1.7 %	
Mean return (ann):		1.7 %	
Volatility:		1.4 %	
Volatility (ann):		1.4 %	

Best period:	1,997,360	4.4 %	(1995-10-31)
Worst period:	0	0.0 bps	(1995-01-31)
Worst drawdown:	2,806,750	5.9 %	(1995-10-31 to 1995-11-30)

Mean size:	991		
Mean equity:	41,518,129		
Mean gross equity:	83,036,258		
Universe turnover:	0		
Turnover:	1,090,700,533		
Mean turnover:	90,891,711		
Mean turnover (ann):	90,891,711		
Holding period (mth):	21.9		

Mean NA weights:	24		
Mean NA returns:	18		

Top/bottom performers by profit:

	id	profit
	1118 45031210	-149164
	505 16961710	-142739
	1704 625064	-118392
	2766 75886F10	-114399
	1387 52465R10	-105515
	2862 81731510	-103997
	997 42331910	145343
	2371 680561	151658
	1546 59511210	153970
	3070 89591910	161162
	2741 74771210	163516
	420 12488310	228322

Top/bottom performers by contribution (%):

	id	contrib
	1118 45031210	-0.30
	505 16961710	-0.27
	1704 625064	-0.24
	2766 75886F10	-0.23
	1387 52465R10	-0.22
	2862 81731510	-0.21
	997 42331910	0.28
	2371 680561	0.30
	1546 59511210	0.30
	2741 74771210	0.32
	3070 89591910	0.33
	420 12488310	0.45

Above, we are given an overview of the entire simulation. In 1995, the StarMine Indicator gave returns of 20.7% when using a long-short strategy over quintiles, and performed best in October. We should take note that the “worst” performing period, or the first month, is artificial because we did not specify any initial holdings, so the simulator assumed that returns were zero for the period. We can also see that the mean returns by period was 1.7%. Interestingly, the mean returns per year is also 1.7%, but this is because we ran the simulation assuming that periods were taken once a year. To adjust this, we would have to re-run the simulation after setting the `freq` slot to 12 as we did above to indicate monthly data collection. Finally, we are given some aggregate statistics on turnover based on the specified initial equity (again, defaulted to 50,000,000 on both long and short sides) and a list of top and bottom performers. Upon examining individual performers and periods, we often wish to more closely inspect a period or a stock to better understand its performance. For example, to view the summary for the worst performing stock, 45031210, we type the following command:

```
> summary(signal.result, id = "45031210")
```

ID statistics for:	45031210

Total Profit:	-75,541
Total Return:	87.0 %
Mean Return:	7.9 %
Mean Return (ann):	7.9 %
Volatility:	26.6 %
Volatility (ann):	26.6 %
Beg Market Value:	-95,164
End Market Value:	NA
Best period:	1995-06-30 78.0 %
Worst period:	1995-10-31 -16.8 %
Trades:	
period shares side	
1 1995-01-31 5144 X	
2 1995-02-28 1307 C	
3 1995-03-31 317 C	
4 1995-04-30 84 X	
5 1995-05-31 667 X	
6 1995-06-30 1917 C	
7 1995-07-31 168 C	
8 1995-08-31 53 C	
9 1995-09-30 94 C	
10 1995-10-31 651 X	
11 1995-11-30 392 C	

From our overall summary, we only know that stock 45031210 was the worst performer by profit, and thus by contribution to the trading scheme. However, we do not know its returns, its market value or even how the simulator traded the stock over the simulation. From above, we see that our “worst” performing stock actually had huge returns - 87%. Immediately, we can see that the simulator shorted this stock heavily, so its strong performance significantly hurt the bottom line. This is concerning, so the typical user would continue to investigate other stocks listed in the worst performing stocks section of the overall summary. Suppose now that we want to further investigate our worst performing security and drill down to its worst performing period, 1995-10-28.

```
> summary(signal.result, period = as.Date("1995-10-31"), id = "45031210")
```

Statistics for:	45031210	1995-10-31

Shares:	-2690	
Trades:	651 (X)	
Returns:	-16.8 %	

The above gives a summary of the stock 45031210 in its worst period 1995-10-28. If the flag `all.data` is set to true, then a summary of a single stock in a period will print all data supplied by the user (including any signal variables which would be very important when we are determining how a strategy has failed in an instance) and generated by `portfolioSim`. As the above summaries show, a by period and by security view can be extremely helpful in determining or understanding performance. After viewing the overall summary, one can easily investigate the most interesting periods and securities, for example the best and the worst, to better gauge the profitability of a trading strategy.

There is one final option for the summary function that allows users to view an overall summary of a single period, rather than the entire simulation.

```
> summary(signal.result, period = as.Date("1995-10-31"))
```

```
Period statistics for:          1995-10-31
-----
Returns:                      4.4 %
Profit:                        NA
Turnover:                      NA
Universe Turnover:            0
Missing Prices:               59
Missing Returns:              50
End Holdings:
-----
```

```
Portfolio: Unnamed portfolio
```

	count	weight	value
Long:	641	1	50,184,093
Short:	641	-1	-50,011,563

```
Top/bottom positions by weight:
```

	id	bps	value
1226	91809610	18	91125
1235	92343610	18	89843
518	50239210	18	89793
1142	83444910	18	89453
464	46631310	17	87029
344	404350	-18	-87710
1146	84760810	-18	-88131
408	444712	-18	-89011
23	010831	-18	-90183
44	02492810	-19	-93600

As we can see, this summary is analogous to the first overall summary, also giving returns and profits. However, we have added a last part that shows the top and bottom positions by weight in the portfolio as of the end of the requested period.

4 Conclusion

Simulating investments over some period in the past is often a time consuming and data-intensive process. The `portfolioSim` package automates the process of running such a simulation, by allowing the user to specify in advance all the details of the portfolio to be maintained, the investment strategy by which to maintain it, and the results from the simulation to be saved. The flexibility of the `portfolioSim` package stems from its interfaces, which can be customized to work with virtually any type of input data and any method of investment. Together with the `portfolio` package, `portfolioSim` provides investors with a powerful and adaptable set of tools for managing their investments and now with the inclusion of the two

introduced convenience functions, and more detailed summary options, investors can use the `portfolioSim` package with even greater ease.