Strategy Tester

Patrick Costello and Matt Johnson

October 11, 2012

We want to develop some system that can handle all types of strategies and all types of data. Following other finance models I think good basic classes for all later projects would be:

- Trade Class. Data will all be non-market,e.g.
  - trade date, trade type notional, bond:coupon, payment frequ, swap: strike, coupfreq. info on what market data type it needs to value.
  - Class functions (members) would be get trade info, price trade(market data input), value trade, calc appropriate greeks for trade type etc
- Market Data Class.
  - Data: raw loaded from bbg/reuters/etc, e.g. bondyields, swaprates, fx, etc
  - DerivedData: swap curves, bond curves, (think we use class inheritence for this e.g have inherited "swap market data", "bond market data" inherited classes)
  - TechnicalData: Stuff we might need for strats, ie moving averages, kalman filters, momentum trackers, etc - again might want to use inheritance
  - Class Members: load data from source to MD object, create derived MD like swap/bond curves, generate technical data from raw

We can then add some more simple classes to combine the above basic classes to produce useful info:

- Portfolio
  - Data: list of trades in portfolio
  - Members: aggregate info function, e.g. portfolio val, greeks, trade history profile etc
- PortfolioSlice - combining Portfolio and MD object at given time value to give all info at that time

- Data: portfolio object, market data object, timeValue
- Members: functions that apply Portfolio/Trade value functions to MarketData object at given time slice to produce Value, greeks, at that time.

So far our classes are pretty generic and versatile - can use outside strategy testing. For strategy testing we define one more class

- Strategy Tester
  - Data: MarketData and Portfolio objects, stratResults
  - Members: signalGenerator - this would generate signal to trade given MD and Portfolio object at each given instant in tiem. E.g momentum tell to buy/sell, mean reversion when to fade, delta hedging how much to hedge
  - Member: updatePortfolio - this would update portfolio given signal
  - Member: run strategy - loop through time applying signaGenerator and update Portfolio to run strategy.
  - Member: Results/Diagnostics, use stratResults to gen info, ie PL, sharpeRatios, TradeProfiles, etc

If we were just looking at one strategy type then the above would be overkill, you could just write a standalone script.

But if we start wanting to look at comparing a bunch of different strategies then we're going to writing a lot of standalone scripts with very similar operatios, loading data, valuing trades, looking at PL and sharpe ratios.

This is what OO is all about - breaking programmes into logical chunks and only specialisng the bits we need to. For strategy testing I think it's the signal generating function. When you work out what this will require the rest of the bits should be easy.

Also a big advantage of moving to OO is you can put in error checking easily - ie you can say what each object needs to work, ie swaptrade objects would need swapcurve objects to values so when you combine into portfolioSlice you could have a check that tells you if you have the right pieces for everything to function. A step we put in the production building stage rather than protyping I think.

Anyway, I've done the above for an example using Moving average crossovers on All ordinary index.