

ALGORITHM BACKTESTING

by Dr John Bates



The term 'backtesting' has been created to describe the process of using past market data as a tool to ascertain how a prospective trading strategy would perform under various circumstances, before that strategy is deployed live in the market. Backtesting algorithms can help to ensure that financial institutions are prepared, as there are a number of requirements, challenges and approaches that should be carefully considered.

Background: Why Algorithmic Trading Needs Backtesting

Algorithmic trading is one of the most discussed topics in capital markets. Initially the term was used to describe the automation of equity execution, i.e. dividing large block trades into slices, using some statistical measure, in order to minimise market impact and achieve a benchmarked price.

However, in the last few years the definition has expanded to include high-frequency trading, i.e. analysing market data in real-time against statistical models in order to detect trading opportunities - and then executing those opportunities. An example of a high frequency algorithm is pairs trading, which monitors correlated instrument pairs, looking for aberrations in the correlated relationship that imply the ability to buy one and sell the other at a profit before they return to correlation.

Algorithmic trading has also spread into asset classes beyond equities, such as futures and options, fixed income and foreign exchange. In each asset class, the same basic principles of monitoring market data for trading opportunities and then automatically executing on the opportunities still apply. However, each asset class differs in the

specific algorithms that are appropriate.

As algorithmic trading has developed, several imperatives have emerged. The first is that you have to identify opportunities first - before your competitors - and build your own custom algorithms to capitalize on opportunities. In many circumstances, you can't rely on pre-built algorithms purchased or leased from vendors or brokers because if everyone has access to the same algorithms, there is a reduced competitive advantage. The second imperative is to gain first-mover advantage by building and deploying algorithms quickly, ahead of the competition. Only those that can quickly deploy are likely to harvest the benefits.

The markets are continually changing and an algorithm that was highly profitable yesterday may not be profitable today. So the third imperative is to continually evaluate the effectiveness of existing algorithms and, if necessary, evolve or decommission them.

Algorithmic trading is both fast moving and highly automated and, as a result, is often associated with increased risk. Many people fear that should things go wrong, they will go wrong so quickly that traders will be unaware and unable to intervene in

time to prevent damage. This leads to concern that the damage may not be constrained and that certain circumstances may cause the entire market to spiral out of control. It is therefore of paramount importance to ensure that an algorithm has been tested under a wide variety of circumstances and in as many trading situations as possible to mitigate such risks.

Backtesting techniques provide a way of evaluating and tuning algorithms for profitability and testing algorithms under various circumstances to ensure they perform as expected in exceptional, as well as normal, trading conditions.

Backtesting Principles, Requirements and Issues

Backtesting uses historical data sequences in order to simulate how an algorithm would have performed if it was trading at a particular point in time. The theory is that by testing it under normal and extreme conditions, performance can be ascertained and sensible responses to exceptional circumstances assured.

The most extreme example would be to test an algorithm with data from both a bull market year and a bear market year. Another example might be

Sample Backtesting Configuration

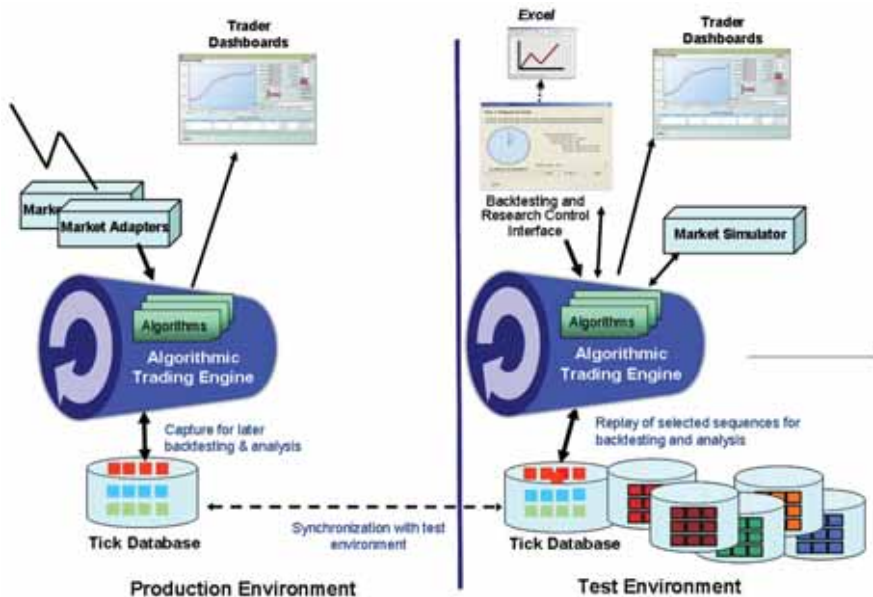


Figure 1. Synchronised data capture in the production and test environments: In the production environment data is sent and received via adapters to various market data feeds and trading venues. This data enters the trading engine and also the tick database for storage. In the testing environment the engine is fed with historic sequences from the tick database and sends trades to a market simulator.

testing a forex algorithm with data from every non-farm payroll day in 2007. The sequences of data selected may be hours, days, weeks, months or years, depending on the requirements of the particular algorithm and backtesting scenario.

Identifying and Backtesting Trading Patterns

In order to create a new algorithm, historical data will most likely be used to research and identify the patterns that an algorithm can use to make money. For example, if a strategist using historical research identifies a particular pattern (e.g. when the moving average of instrument X exceeds the price by threshold Y, the price will always rise by Z), an algorithm can be created to capitalise upon this opportunity.

Historical data, in which the pattern was originally spotted, can then be used to backtest and evaluate whether the prospective algorithm would identify all relevant opportunities and take advantage of them.

Acquiring and Managing Years of Data

In order to backtest algorithms, a store of relevant historic data must be kept. In equities, for example, this could mean tick and quote data, with full depth of book, going back ten years. Where an institution is trading across borders, there may be a requirement to store data from exchanges in the US, UK, Canada, Mexico and Japan. Depending on the detail of data that is stored, there could be several thousand market events per second on an individual exchange - equating to millions per day, and even billions per year. In data storage terms this equates to several terabytes per year.

This data has to be captured, stored, indexed and queried to support efficient backtesting. One way of acquiring the data is to buy it from trading venues or data vendors. However, such data is often delivered on CDs after-the-fact. In-house data capture is required to test with today's and yesterday's data. Market data has a temporal dimension

in which the sequence of data is material to understanding what happened. This temporal dimension must be stored and indexed as a first class property, so it can be replayed in order.

High Performance Backtesting

Dealing with data volume is not the only issue in backtesting. In the quest for ever-faster deployment of new algorithms, there is a strong desire to backtest algorithms quickly. Traditional databases are not fast enough to capture or replay - in real-time - the volume of events needed to support trading usage and are not designed to handle time-series (temporally ordered) data.

A new breed of tick database has evolved to fulfil these requirements. These databases support the temporal ordering of events and can replay market data in the same order as the events originally happened. Such databases can also handle storage and replay of thousands of events a second. In a high performance backtesting framework, it may be possible to run many thousands of algorithmic permutations against historic sequences at the same time. This enables thousands of possibilities to be evaluated concurrently and hugely accelerates time-to-market for algorithms.

Backtesting Multiple Asset Classes

Algorithms for other asset classes may introduce additional requirements to the backtesting operation. For example, foreign exchange traders may want to backtest using data from multiple trading venues e.g. EBS, Reuters, Currenex, Hotspot and a bank's own liquidity pools. In futures and options it might be CBOT, CME, Eurex and Liffe, while in fixed income it might be Brokertec and eSpeed.

In addition, some algorithms are cross-asset in nature, trading multiple asset classes in the same strategy. As a result, multiple asset-class streams must be replayed in order to backtest and it may be necessary to replay a sequence composed of several independent →

Sample Backtesting Interface

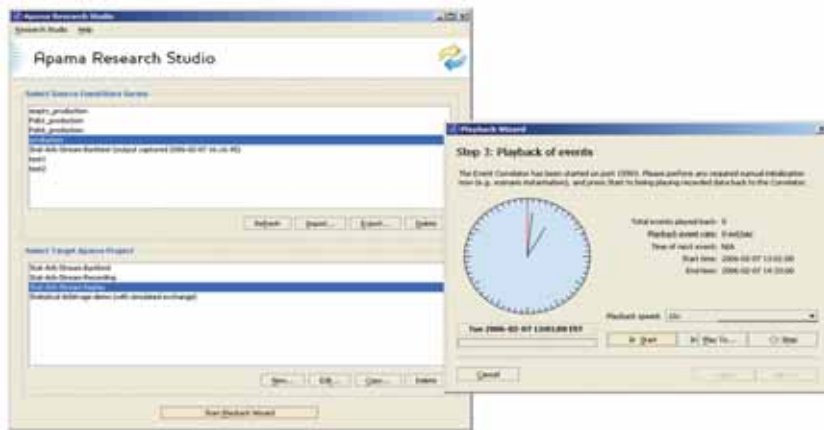


Figure 2. Backtesting Control Panels: On the left screen a historical sequence and a strategy to stream it through has been selected for backtesting. On the right screen a wizard guides the user through control options including speed of playback, pause and step controls.

asset-class specific sequences, retaining the temporal ordering and relationship of the different market streams.

Backtesting with News

A new backtesting requirement that has emerged over the last year is the need to support algorithms that trade on news. Certain news moves the market, particularly when it is economic news or shock news (e.g. an unexpected war or hurricane). Traders realise that if they can respond before their competitors they can gain an advantage and are using algorithms to monitor news along with other market data.

As an example, Dow Jones has made this process more quantitative by creating elementized news feeds, which use tags to identify specific elements within news events. In order to backtest algorithms that incorporate news, news feed events have to either be recorded or simulated. Since elementized news feeds are a new phenomenon, it has not been possible to acquire years of elementized news, but as algorithms that trade on news become more commonplace more archives of unstructured and elementized news will become available for backtesting.

Simulating Market Impact

The most complex requirement in a backtesting framework is the challenge of simulating market impact. Replaying historical data and feeding it into an algorithm is one thing. But what about simulating the circumstance where an algorithm wants to take advantage of opportunities in the market? Where does it place its orders? In a comprehensive backtesting framework a market simulator is required. Algorithms can route orders to the market simulator, which will respond as an external market would (such as an equities exchange, futures exchange, forex venue or bond venue).

The complexity of simulating market impact begins when you consider that when backtesting with data from the past, your algorithm wasn't actually there, and thus its actions will not have an impact on the historical data. If you hit a bid in the historic data, by default that bid will still be there. This issue can never be fully addressed without creating a time machine that could go back to the day in question and run the algorithm live. However, there are certain techniques that enable more realistic impact simulation. One such technique is to use a simulator that can remember

deltas. Deltas describe cumulative changes to the historic data. For example, if our algorithm hits a bid or offer, although in the past it was still there, the simulator should remove it from the order book and remember this as a delta to be applied henceforth to history.

In other circumstances, traders may want to simplify things and assume perfect liquidity in order to test certain rules in an algorithm. Alternatively, they may want to use a totally simulated world that uses synthetic market data rather than historic market data.

Strategy Tuning

A key aspect to backtesting is strategy tuning; running strategies in various different configurations, with the same data, to see which permutation is the most profitable. Data on each algorithmic permutation can be collected and compared with the most profitable configuration to be used for live trading. The effectiveness of a particular permutation may change over time and thus regular tuning is required to ensure the algorithm is being run in its optimal configuration.

Conclusion

To better ensure that algorithms are ready for any eventuality and will actually work as expected requires high performance time-series capture and replay, large data storage, realistic market simulation and continuous algorithm tuning. The latest backtesting approaches, including high performance tick databases for capture and replay of time-series data can help ensure there are no surprises. Those that use backtesting appropriately will be well prepared to earn their 'Boy Scout' algorithmic trading badge.

John Bates is Founder and Vice President, Apama Products, Progress Software