AlgoTCA - GSoC 2019

by Vito Lestingi, Jasen K Mackie, Brian G Peterson

Abstract As part of the R organization's participation in the Google Summer of Code 2019 we replicate, and to some degree extend, the pre and post-trade TCA models as disclosed in "The Science of Algorithmic Trading and Portfolio Management" – Robert L. Kissell. The main focus of this 3-month programme was to build an Implementation Shortfall TCA model and a range of benchmark price performance models (VWAP, PWP, RPM, Arrival Cost etc), models for statistically comparing different algorithmic execution performances and lastly replicating Kissell's I* model for Market Impact estimates. Using 11 months of JSE data and 16200 observations we are able to train a non-linear least squares estimation function which converges on a plausible solution, allowing us to validate the model which can be used in production for larger datasets with presumably lower variance.

Introduction

Post-trade Transaction Cost Analysis (TCA) is critical for assessing the efficiency with which trading decisions are implemented, especially for the institutional investor whose trade ideas can span multiple time periods to implement and which risks adverse market impact if not managed accordingly. Pre-trade and intra-day TCA is an equally vital component for making optimal trading decisions based on market conditions and trading constraints. The blotter package in R is an ideal package in the R ecosystem for hosting TCA functionality since it serves as the transaction infrastructure for building and analysing simulated and production trading systems. In this article we describe the functions implemented as part of this GSoC 2019 project in 3 parts, namely: 1. Post-trade TCA models, 2. Models for the statistical comparison of algorithmic executions, 3. Pre-trade Market Impact estimation using the I* model as documented in "The Science of Algorithmic Trading and Portfolio Management" – Robert L. Kissell, originally developed by Kissell and Malamut in 1998. Where appropriate we include examples, elaborate on extensions, document the journey of a supposed analyst and communicate results and findings relevant to a user. Lastly we comment on interesting paths of future work.

1. Post Trade TCA

Implementation Shortfall

Implementation Shortfall is a measure that represents the total cost of executing an investment idea. Implementation Shortfall is calculated as the difference between the paper return of a portfolio where all shares are assumed to have transacted at the manager's decision price and the actual return of the portfolio using actual transaction prices and shares executed.

Implementation Shortfall can be implemented in 4 ways:

1. Assuming Complete Execution, implying a zero opportunity cost component. This is the default method for `Implementation Shortfall`

The simplest formulation for the Complete Execution IS method is:

$$IS = S.P_{avg} - S.P_d + fees$$

Note that we add fees, as a positive metric indicates a cost.

2. Using an Opportunity Cost (Perold 1988) component, where not all shares originally allocated for trading are finally executed. Opportunity Cost is the cost of not executing a portion of the originally allocated shares `S` for execution. This could be due to limit price constraints or a lack of liquidity.

The formulation for Opportunity Cost is:

$$(S - \sum s_j).(P_n - P_d)$$

The Implementation Shortfall formulation of Perold (1988) can be written as:

$$IS = \sum s_j.(P_{avg} - P_d) + (S - \sum s_j).(P_n - P_d) + fees$$

3. Expanded Implementation Shortfall (Wayne Wagner)

Wayne Wagner's implementation categorizes costs into delay, trading and opportunity related costs. Assuming \$ P_{d} \$ is the decision price, \$ P_{0} \$ is the price when trading begins (ideally Arrival Price, defined as the mid-price at order arrival, alternatively Last Price at order arrival or failing that data availability then first transaction price), and \$ P_{n} \$ is the price at the end of trading. The Expanded IS can be written as:

$$(P_n - P_d) = (P_n - P_0) + (P_0 - P_d)$$

If you substitute the RHS into Perold's IS, then IS can be written as:

$$IS = (\sum s_j p_j - \sum s_j P_d) + (S - \sum s_j).((P_n - P_0) + (P_0 - P_d)) + fees$$

This formula can be re-written into a separate delay, trading and opportunity cost related component as follows:

$$ExpandedIS = S(P_0 - P_d) + (\sum s_j)(P_{avg} - P_0) + (S - \sum s_j)(P_n - P_0) + fees$$

where each term (excluding fees) reflects the delay, trading and opportunity cost components respectively. Wagner's method allows for an additional decomposition of the "Delay related cost" into the *opportunity delay cost* component and the *trading delay cost* component. Our implementation provides this breakdown in the output.

4. Market Activity IS, which assumes the analyst is unaware of the manager's decision price. This method is equivalent to the Wagner formulation except that the first term is excluded in order to assess only market activity IS:

$$MarketActivityIS = (\sum s_j)(P_{avg} - P_0) + (S - \sum s_j)(P_n - P_0) + fees$$

Implementation Shortfall - Examples

For the IS examples we borrow from the code in the function help documentation. The first 2 examples below illustrate the output for the Complete Execution and Market Activity implementations using identical datasets. The Implementation Shortfall quantums are identical as we assume complete execution in both instances including the Market Activity IS, although you will notice this version returns additional info when compared with the Complete Execution version, including filled and unfilled units, the opportunity cost component and fees.

```
### Complete Execution IS
impShortfall("testport", "test_txns",
          paQty=5000,
          priceStart=10,
          priceEnd=11,
          arrPrice=10,
          method='Complete')

#>      Symbol   Method Paper.Ret Actual.Ret Shortfall
#> 1 test_txns Complete      5000       2400      2600

### Market Activity IS
impShortfall("testport", "test_txns",
          paQty=5000,
          priceEnd=11,
          arrPrice = 10,
          method='Market')

#>      Symbol Method t.Txn.Qty u.Txn.Qty Trade.Cost Opp.Cost Fees Shortfall
#> 1 test_txns Market      5000         0       2500        0  100      2600
```

The next 2 examples illustrate the Perold and Wagner versions, respectively, again using identical datasets in both cases. We assume only 4,000 of the originially allocated 5,000 units eventually trade.

For the Perold output, we see the breakdown between Execution Cost, Opportunity Cost and Fees separately.

```
### Perold
impShortfall("testport", "test_txns",
          paQty=5000,
          priceStart=10,
          priceEnd=11,
          arrPrice=10,
          method='Perold')

#>      Symbol Method t.Txn.Qty u.Txn.Qty Exe.Cost Opp.Cost Fees Shortfall
#> 1 test_txns Perold      4000      1000     2000     1000   80      3080
```

In the case of the Wagner implementation, we see the extra breakdown compared with Perold, ultimately equating to the same hypothetical Implementation Shortfall measure.

```
### Wagner
impShortfall("testport", "test_txns",
          paQty=5000,
          priceStart=10,
          priceEnd=11,
          arrPrice=10.25,
          method='Wagner')

#>      Symbol Method t.Txn.Qty u.Txn.Qty Opp.Delay Trade.Delay Delay.Cost
#> 1 test_txns Wagner      4000      1000       250        1000       1250
#>   Trade.Cost Opp.Cost Fees Shortfall
#> 1       1000      750   80      3080
```

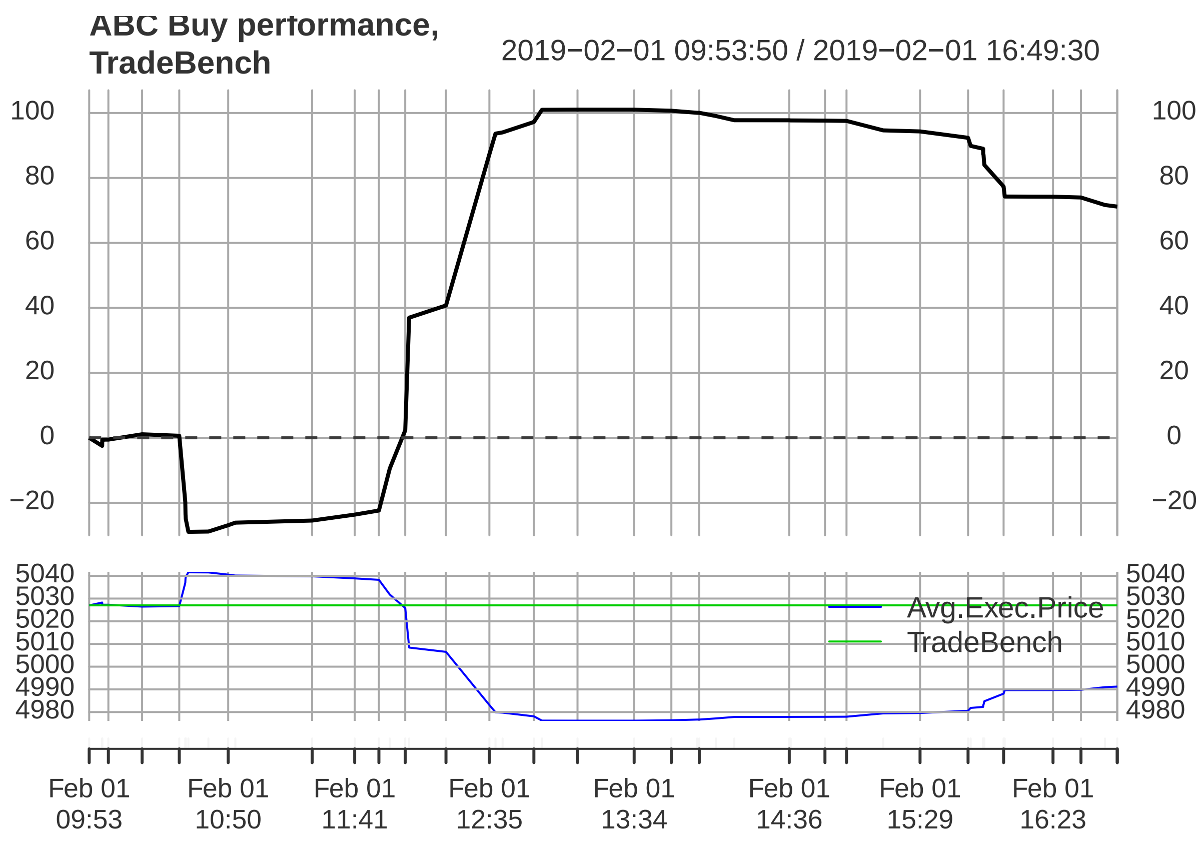
Benchmark Price Performance - Examples

We built a single function for computing TCA benchmarks where the benchmark price is one of: Arrival Price, Day's Open/Close/Other, Participation Weighted Price (PWP), Volume Weighted Average Price (VWAP) or a qualitative score with the Realtive Performance Measure (RPM). When all methods are used in conjunction, an analyst can derive measures for each benchmark giving a different view of the resultant execution. In addition to giving an overall score, we have built an S3 method for plotting the performance through the life of the execution which allows an analyst to identify specific time periods that warrant additional review.

benchmark='TradeBench'

For all 4 methods we use the same sampled set of public trades data to simulate a private execution. In all cases, a positive value indicates outperformance and a negative value, underperformance. In the case of benchmark='TradeBench' the benchmark price is the first transaction price in the execution, as a proxy for the Arrival Price. For a true Arrival Cost benchmark using the actual Arrival Price, the user can use benchmark='MktBench' with the Arrival Price specified for with the 'priceToBench' argument.

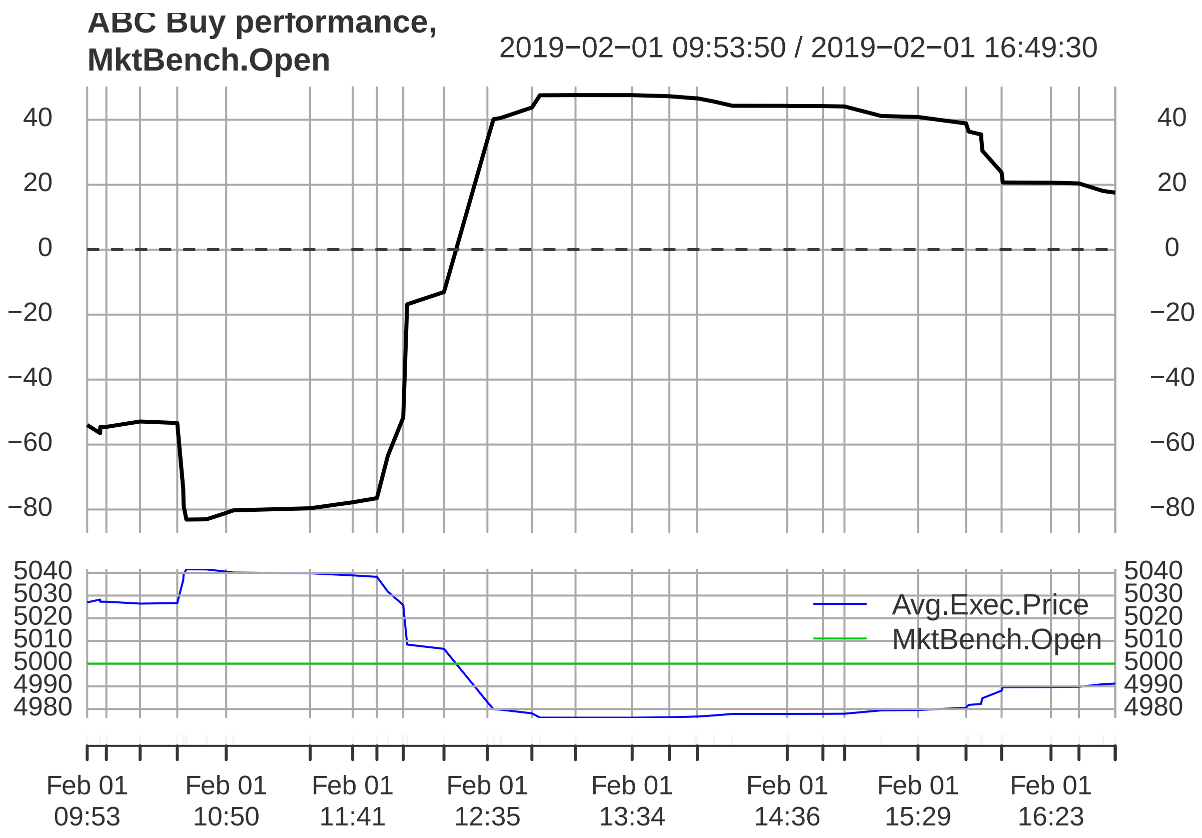
```
benchTradeBench <- benchTradePerf('abc.port.day', 'ABC', side = 1, benchmark = 'TradeBench', MktData = ABC.da
last(benchTradeBench$Trades.TradeBench.Perf)

#>                  Dates Symbol Side Avg.Exec.Price TradeBench Performance
#> 50 2019-02-01 16:49:30    ABC  Buy       4991.218       5027    71.17988

plot(benchTradeBench, benchmark = 'TradeBench')
```



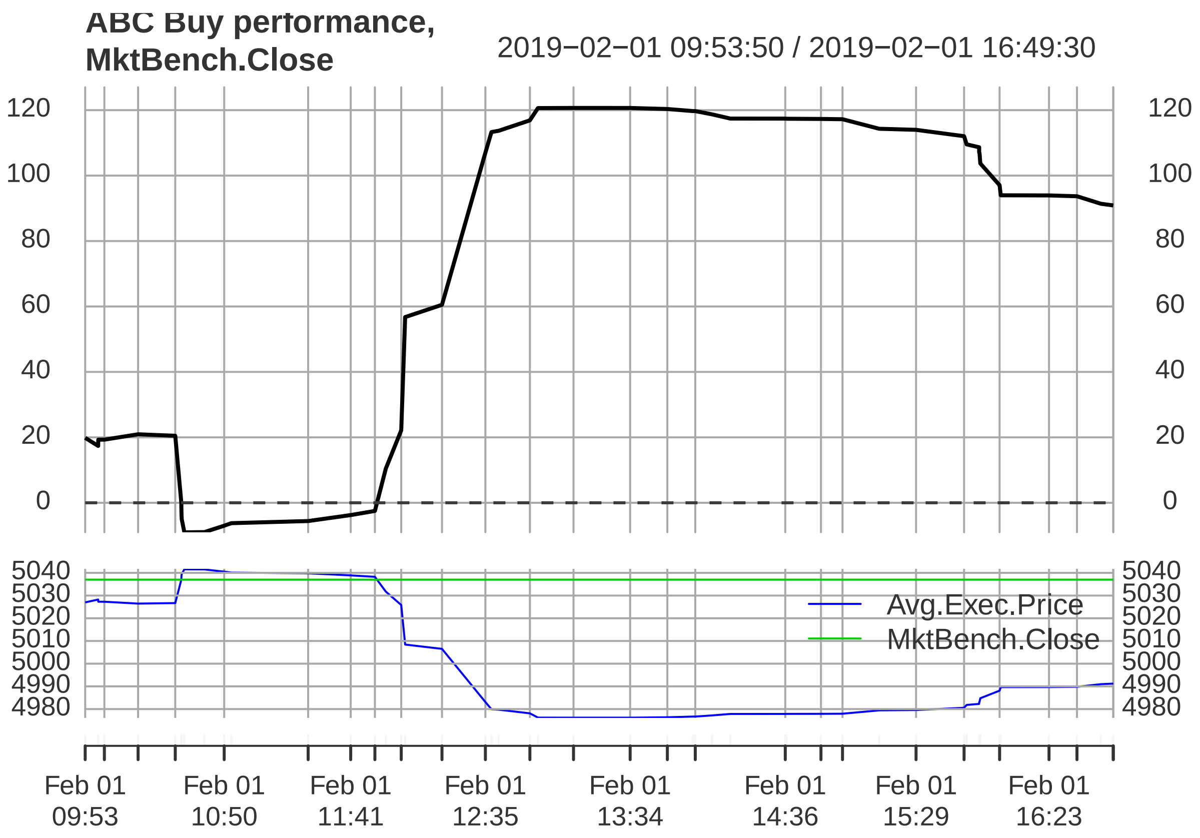
benchmark='MktBench'

As alluded to previously, with benchmark='MktBench' the user can specify "Open", "Close" or a numeric price value for benchmarking.

```
benchMktBenchOpen <- benchTradePerf('abc.port.day', 'ABC', side = 1, benchmark = 'MktBench', type = list(pric
last(benchMktBenchOpen$Trades.MktBench.Perf)

#>                    Dates Symbol Side Avg.Exec.Price MktBench.Open
#> 50 2019-02-01 16:49:30    ABC  Buy       4991.218          5000
#>    Performance
#> 50    17.56425

plot(benchMktBenchOpen, benchmark = 'MktBench')
```



```
benchMktBenchClose <- benchTradePerf('abc.port.day', 'ABC', side = 1, benchmark = 'MktBench', type = list(pri
last(benchMktBenchClose$Trades.MktBench.Perf)

#>                   Dates Symbol Side Avg.Exec.Price MktBench.Close
#> 50 2019-02-01 16:49:30    ABC  Buy       4991.218           5037
#>    Performance
#> 50    90.89166

plot(benchMktBenchClose, benchmark = 'MktBench')
```



benchmark='VWAP'

A widely used benchmark is the Volume Weighted Average Price (VWAP) benchmark. The benchmark is defined as:

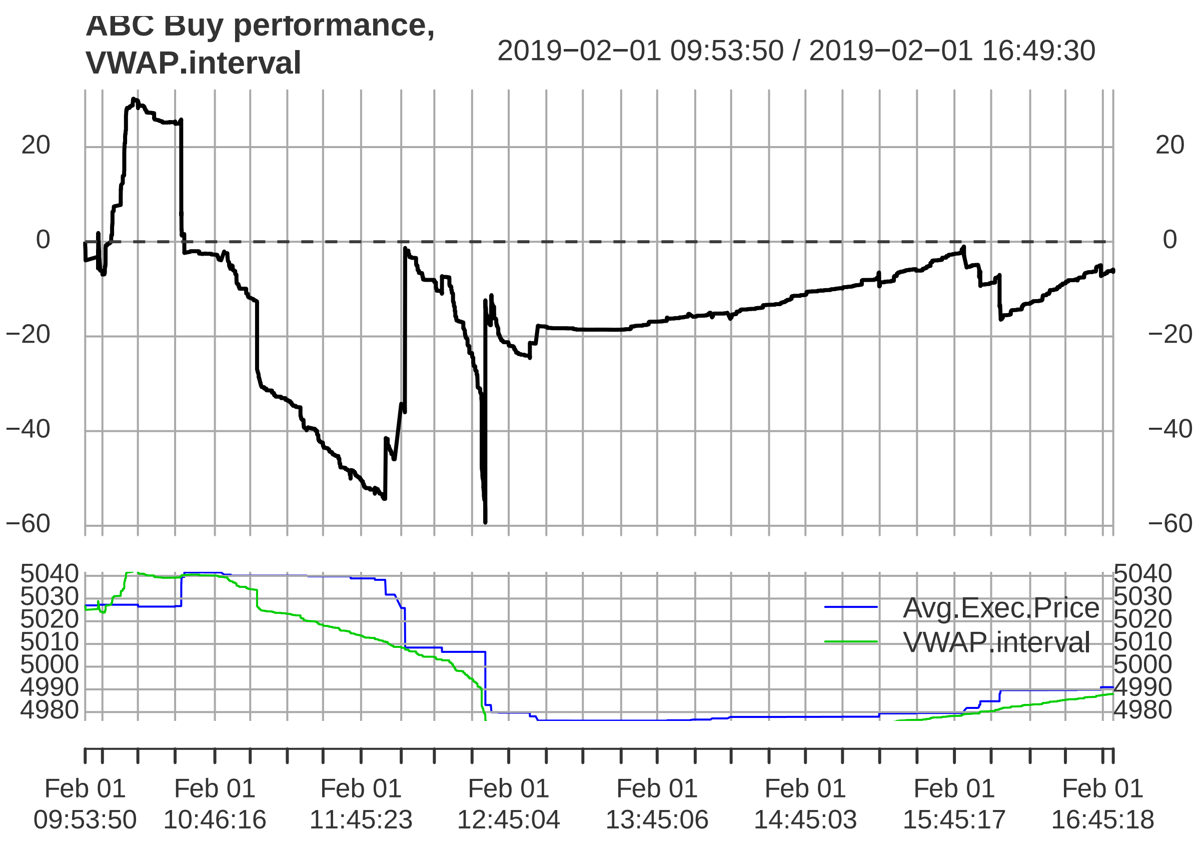
$$VWAP = \frac{\sum P_j Q_j}{\sum Q_j}$$

P_{j} is the market price and Q_{j} the market volume, during j trading periods activity of the market. Two different types of VWAP benchmarks are included in the present function, the Interval VWAP and the Full VWAP. Referring to the former as the VWAP where the j market trading periods considered are the ones during which the order is being executed, whereas the latter includes all the j market periods from order execution beginning to last transaction. The VWAP benchmark varies by timespan considered and is commonly used as a proxy for fair market price. It can differ by data vendors specific market data filtering. There are recognized drawbacks of this benchamrk. First of all, the larger the order the closer the execution will be to VWAP. Second, where large block trades occur these could skew the benchmark. Lastly, it is not an indicated comparison across stocks or different days for the same stock.

```
benchVWAPinterv <- benchTradePerf('abc.port.day', 'ABC', side = 1, benchmark = 'VWAP', type = list(vwap = 'ir
last(benchVWAPinterv$Trades.VWAP.Perf)
```

```
#>                    Dates Symbol Side Avg.Exec.Price VWAP.interval
#> 7091 2019-02-01 16:49:30    ABC  Buy       4991.218      4988.044
#>      Performance
#> 7091   -6.363777
```

```
plot(benchVWAPinterv, benchmark = 'MktBench')
```



benchmark='PWP'

A variation of the VWAP benchmark is given by the Participation Weighted Price (PWP) benchmark, where the weighting is with respect to the PWP shares:

$$PWPshares = \frac{Tradedshares}{POV}$$

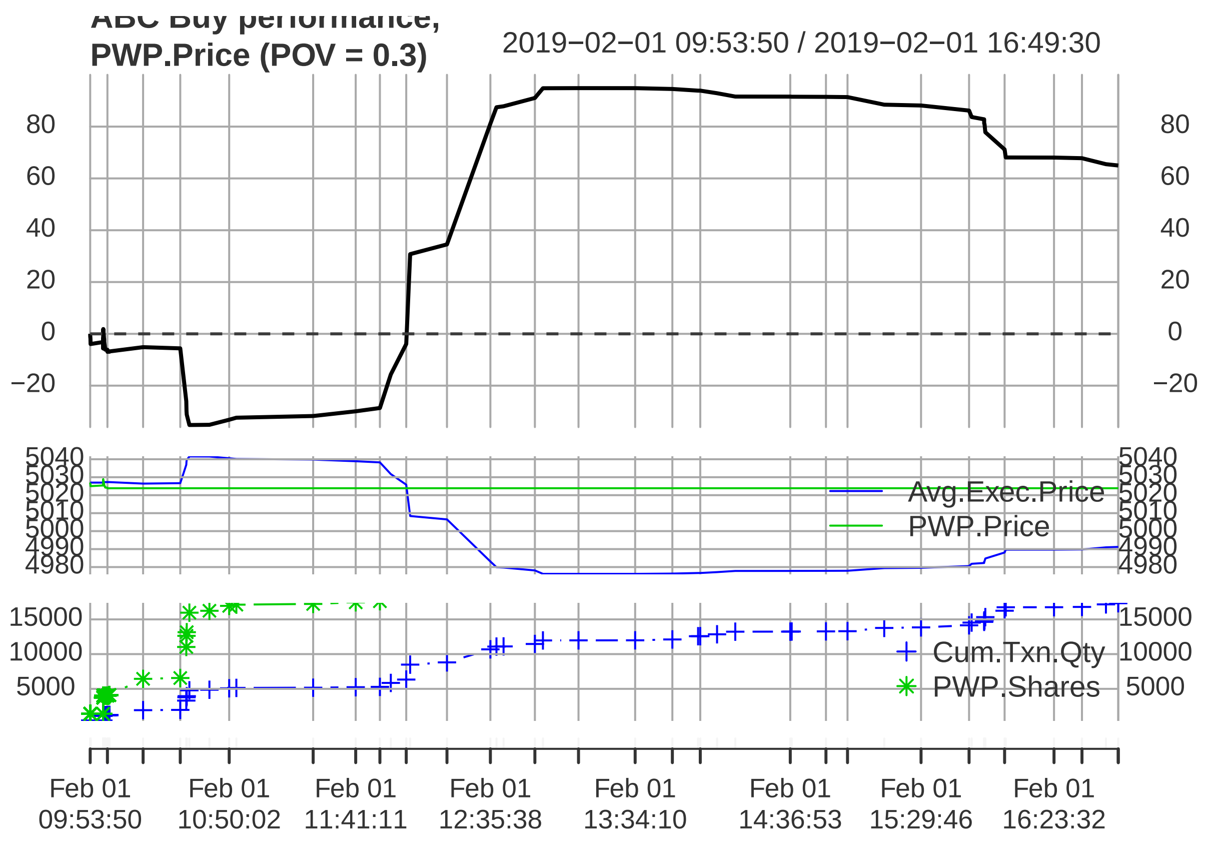
POV refers to the percentage of volume.

The PWP benchwark is:

$$PWPprice = \frac{\sum P_h Q_h}{\sum Q_h}$$

where h are the periods from the arrival time of the order into the market until when the PWP shares are completely executed. As the VWAP, the PWP benchmark provides a glimpse into market fair prices. However this benchmark have limitations similar to the VWAP. It is subject to manipulation in that the market price can be kept inflated by larger orders. Furthermore, as the VWAP, it is not comparable between stocks or across days for the same stock. Also, the benchmark may be biased by temporary impact dissipation.

```
benchPWP <- benchTradePerf('abc.port.day', 'ABC', side = 1, benchmark = 'PWP', POV = 0.3, MktData = ABC.day)
last(benchPWP$Trades.PWP.Perf)

#>                     Dates Symbol Side Cum.Txn.Qty POV PWP.Shares
#> 164 2019-02-01 16:49:30    ABC  Buy       17322 0.3      57740
#>     Avg.Exec.Price PWP.Price Performance
#> 164       4991.218  5023.869    64.99206

plot(benchPWP, benchmark = 'PWP')
```



benchmark='RPM'

Lastly, the Relative Performance Measure (RPM), which differs from the PnL metrics above, is a percentile ranking of trading activity. Its expression depends on the side of the trade:

$$RPM_{buy} = 0.5 * \frac{Totalvolume + VolumeatP > P_{avg} - VolumeatP < P_{avg}}{Totalvolume}$$

$$RPM_{sell} = 0.5 * \frac{Totalvolume + VolumeatP < P_{avg} - VolumeatP > P_{avg}}{Totalvolume}$$

where P is the market price specified. The an RPM over 50% is considered as an indication of superior trades, more precisely the RPM can be mapped to a qualitative score of the trades:

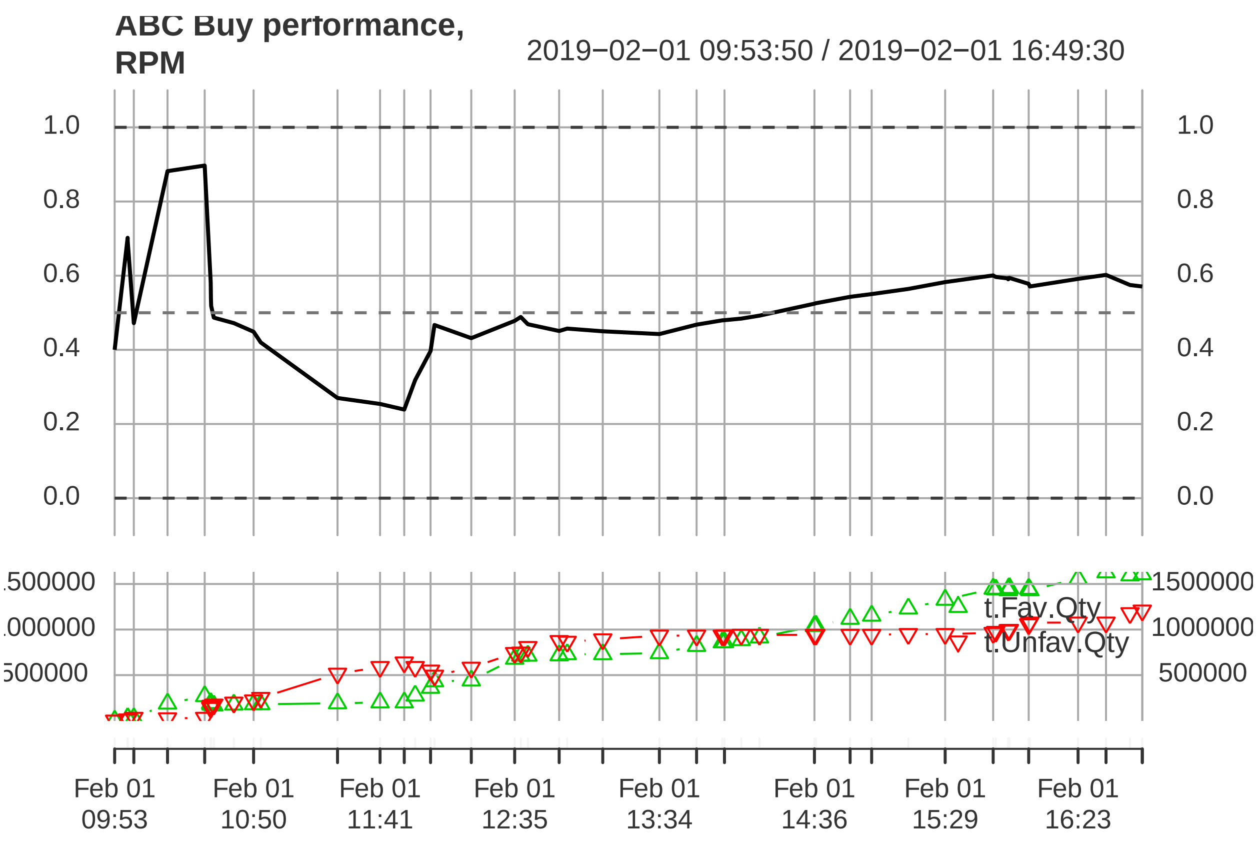
0 <= RPM < 20 Fair 20 <= RPM < 40 Poor 40 <= RPM <= 60 Average 60 < RPM <= 80 Good 80 < RPM <= 100 Excellent

This measure is considered as preferred to the VWAP metric because it overcomes some of its drawbacks: it can be used to compare performance across different stocks, days, and volatility; it is not less influenced by large blocks trade at extreme prices.

```
benchRPM <- benchTradePerf('abc.port.day', 'ABC', side = 1, benchmark = 'RPM', MktData = ABC.day)
last(benchRPM$Trades.RPM.Perf)
```

```
#>                   Dates Symbol Side Avg.Exec.Price Mkt.Price t.Mkt.Qty
#> 50 2019-02-01 16:49:30    ABC  Buy       4991.218      5012   2814601
#>    t.Fav.Qty t.Unfav.Qty      RPM Quality
#> 50   1606594     1208007 0.570807 Average
```

```
plot(benchRPM, benchmark = 'RPM')
```



This section may contain a figure such as Figure 1.

Figure 1: The logo of R.

Another section

There will likely be several sections, perhaps including code snippets, such as:

```
x <- 1:10
x

#>  [1]  1  2  3  4  5  6  7  8  9 10
```

Summary

This file is only a basic article template. For full details of *The R Journal* style and information on how to prepare your article for submission, see the Instructions for Authors.

Vito Lestingi
Student - University of Rome
line 1
line 2
author1@work

Jasen K Mackie
Algorithmic Trading Services - Iress
line 1
line 2
jasen.mackie@iress.com

Brian G Peterson
TODO
line 1
line 2
brian@braverock.com