

# TradeCosts

Jeff Enos, David Kane, Arjun Ravi Narayan, Aaron Schwartz, Daniel Suo, and Luyi Zhao

## Introduction

In the discussion of trading, trade costs are often not given enough attention. For example, each trade is associated with an execution cost and an opportunity cost. If these costs are not considered, a portfolio may underperform its projected returns.

The **tradeCosts** package aims to calculate and report the following trade costs: price appreciation, market impact, and opportunity cost. However, in this iteration we only fully implement price appreciation trade costs. We follow the definitions in Kissel and Glantz (2003).

## A trading example

Consider a single trade of a single stock. On the evening of August 1<sup>st</sup>, a trader decides to purchase 10,000 shares of IBM at \$10 a piece. We define this price as the “decision price” of the trade. The trader makes this decision after the market has closed, however, so his broker cannot buy the shares. The next day, the broker tries to buy the 10,000 shares at the best price he can get, but the price rises, and the broker buys the shares at \$11 each. We define this price as the “execution price”. The trade does not go well — the trader pays \$11,000 when he expected to pay \$10,000 for a total trade cost of \$1,000 or 9.1% of the execution price.

Simply, trade costs are calculated by comparing the execution price with a benchmark price. In the above scenario, the benchmark price is the decision price. Here lies the complexity of trade cost calculations. Depending on purpose, one can choose from many different benchmark prices. A decision price benchmark might measure broker efficiency, but one could also measure broker efficiency by comparing the execution price against the day’s volume-weighted average price, or VWAP (Wikipedia (2007)). This comparison would measure the broker’s purchase price relative to the average purchase price of all others in the market that day. If VWAP on August 2<sup>nd</sup> were \$10.50 and the trader uses this price as his benchmark, then the total trade cost is \$500. The remaining \$500 dollars between the benchmark and decision prices is attributed to price appreciation.

From this example, it is clear that trade costs cannot be ignored. It is just as important to find and lower trade costs as it is to improve research and modelling.

## tradeCosts architectural overview

The **tradeCosts** package requires at least two data frames. One data frame must contain all of the trading data with columns representing period, id, execution quantity, execution price, and side (“B” for a buy, “S” for a long sell, “X” for a short sell, and “C” for a cover). The other data frame, called the dynamic descriptions data frame, must include price and benchmark data for at least each stock in each period specified in the trade data frame. These two data frames are then merged and the execution price is compared to the benchmark price. An optional third data frame containing static data can also be included to prevent repeating unchanging data. A company will most likely keep the same sector, for example, from period to period.

**tradeCosts** offers a function called `trade.costs` that takes these data frames and generates the trade cost reports.

## A single period trade cost calculation

Let us use the first period of the sample data included with **tradeCosts**. Below is the trade data for March 1<sup>st</sup>, 2007 with the required columns and column names. Note that we are using a subset of the sample data included.

	period	id	side	exec.qty	exec.price
5	2007-03-01	03818830	X	60600	1.60
8	2007-03-01	13959410	B	4400	32.21
9	2007-03-01	15976510	X	13600	7.19
10	2007-03-01	22122P10	X	119000	5.69
11	2007-03-01	25383010	X	9200	2.49
13	2007-03-01	32084110	B	3400	22.77
49	2007-03-01	88362320	X	31400	3.01
54	2007-03-01	98385710	B	6400	11.80

Next, we show some of the dynamic data:

	id	period	price	vwap	prior.close
19	00797520	2007-03-01	3.85	3.88	3.34
29	010015	2007-03-01	1.27	129.35	2.53
75	023282	2007-03-01	6.13	613.57	12.02
118	03818830	2007-03-01	1.62	1.58	1.62
136	047628	2007-03-01	2.85	285.67	5.61
226	091139	2007-03-01	4.21	418.48	8.22

And finally some of the the static data:

id	symbol	name	sector
90273710	UCI	Uici	FIN
45071R10	XXIA	Ixia	TEC
67019E10	NST	Nstar	UTL
88579Y10	MMM	3m Co	IND
98302410	WYE	Wyeth	HTH
72581110	PIXR	Pixar	CND

We can now analyse our trade costs using the `trade.costs` function with the three data frames as arguments:

```
> result <- trade.costs(trade.mar.1.2007,
+   dynamic = dynamic.mar.1.2007,
+   static = static.mar.2007)
```

The `trade.costs` function also has a number of optional arguments. The benchmark price is defaulted to VWAP, and the price compared to the benchmark price defaults to the execution price. We will discuss the optional argument `batch.method` and `num.trades` later. Note that whatever variable provided as the benchmark should be in either the trade or dynamic data frame.

The `trade.costs` function creates a `tradeCostsResults` object. Let us look at the trade costs reported.

```
Trade cost analysis with benchmark price: vwap
Total Market Value:      1,283,963
First Period:           2007-03-01
Last Period:            2007-03-01
Trade Costs:            -6,491
```

By calling the `show` method, we see that a small statement of the results is provided. There is a report of the total market value of all the trades executed. We are most concerned with the last figure, which is the report on the trade costs. Note that the trade costs here are negative. This is entirely possible, if the execution price is lower than the benchmark price. A more in-depth report is provided by the `summary` method, which we shall look at later.

## Multi period trades

Trade costs, when measured against VWAP, show performance against the “going rate”. But when we want to measure trade costs against the decision price we must make a careful distinction. The decision price might stay the same over many periods of trading, while benchmarks such as VWAP vary each period.

Suppose now that the trader decides to buy a stock on a particular day, but he wants to buy so many shares that it takes several days to complete the trade. For example, consider a purchase of 6 million shares of Lockheed Martin (NYSE:LMT). The average daily traded volume of Lockheed Martin is around 3 million shares. Assuming the trader’s broker can buy half of all the shares put up for sale each day, i.e. 50 percent of the daily traded volume, it would take approximately 4 days to complete the trade. If we specified the `benchmark.price` as `prior.close`, the benchmark price would be recalculated each evening as the price prior to closing. However, the true benchmark price is actually the

price prior to closing on the first evening, when the decision to buy was made.

The solution to this problem is to correct for this problem is to “batch” the trades. After ordering all trades by period, `tradeCosts` consider any consecutive string of trades on one side — buy, sell, cover, short — to be one batch. For example, if we buy a stock one day and the next trade we make on that stock is another buy, this string is considered one batch. This can occur on consecutive periods, or in between an arbitrarily large number of periods. However, if we buy a stock on one day, sell it the next, and buy it again, we have created three batches.

The `trade.costs` function takes a parameter called `batch.method` that can take one of two values. The default, `unique`, treats each trade as a separate batch. `same.sided` treats consecutive trades of the same side as a single batch.

## A summary of the report

Now let us look at a summary of the report produced from the result of the `trade.costs` function. Calling the `summary` method gives us three sections in addition to the short report accessible by the `show` method. We can also specify the number of trades, coded as `num.trades`, to show in our report summary of best and worst trades. Let us look at a summary of the result object we created earlier:

```
> summary(result, num.trades = 3)
```

### Trade Cost Analysis

Benchmark Price: vwap

#### Summary Statistics

```
Total Market Value:      1,283,963
First Period:             2007-03-01
Last Period:              2007-03-01
Total Exec. Cost:         -6491.3
```

#### Best and worst batches over all periods

	exec.qty	cost	decision
1	119,000	-3,572	4
2	60,600	-1,615	1
3	31,400	-1,235	7
6	9,200	33	5
7	4,400	221	2
8	3,400	370	6

#### Best and Worst IDs

	id	exec.qty	cost
1	22122P10	119,000	-3,572
2	03818830	60,600	-1,615
3	88362320	31,400	-1,235
6	25383010	9,200	33
7	13959410	4,400	221
8	32084110	3,400	370

NA Report

```

                count
side            1
id              0
base.price     0
exec.qty       0
period         0
benchmark.price 0

```

The first section is the same as that produced with the `show` method. Following that, we see a report of the best/worst batches. In this example, since we have used the unique batching method, we do not see any new information. When trades are batched using the `same.sided` method, the summary will provide additional reports on each batch.

The third section shows the best and worst `id`'s and their respective trade costs. That is, which stocks were the most expensive, and which were the most efficient.

The final section shows a count of the NA values in the data. An NA value represents missing data, and this is reported and NAs in the base or benchmark prices prevents trade cost calculations for the corresponding `ids`. This does not mean the trade costs were zero, however, so a high number of NAs should lend reason for further investigation.

## An example using `same.sided` batching

Looking at the summary report above, the first report shows a list of batches. To run a trade cost analysis with the `batch.method` parameter as `same.sided`, we would specify:

```

> result.batched <- trade.costs(trade.mar.2007,
+   dynamic = dynamic.mar.2007,
+   static = static.mar.2007,
+   batch.method = "same.sided")

```

Note that we specify the benchmark as `decision.price`. This is a special mode which tells the `trade.costs` function to calculate the decisions based on the `prior.close` variable.

Let us now run through an example where we show the difference between a batched and unique trade cost calculation for a single stock.

In the `trades.mar.2007` dataset, we take the stock `CNTY`, which contains the following set of trades:

```

    period      id side exec.qty exec.price
2 2007-03-09 15649210 C    16100      9.90
3 2007-03-15 15649210 X    16000      9.33
4 2007-03-19 15649210 X    42400      8.73
5 2007-03-20 15649210 X    15400      8.65

```

If we examine the batches for both trade cost calculations, we see that the "unique" method gives us:

```

    id side exec.qty execution.cost cost
1 15649210 C    16100      -483  -483
2 15649210 X    16000       4160  4160
3 15649210 X    42400      33072 33072
4 15649210 X    15400       -924  -924

```

The same sided batched method gives us:

```

    id side exec.qty execution.cost cost
1 15649210 C    16100      -483  -483
2 15649210 X    73800      50054 50054

```

From the above examples, we can see that the `trade.costs` function can be tailored to the user's needs. If the benchmark price is VWAP, it may make more sense to use the unique batch method because VWAP changes with each period. However, if the benchmark price is the decision price, the `same.sided` method is more appropriate.

## Plots

The `tradeCosts` package includes a `plot` function that displays bar charts of the trade costs. It requires two arguments — the `texttttradeCostsResults` object, and a string that describes the type of plot to create.

The simplest plot is a time series of total trade costs in basis points over each period. To call this type of plot, we provide the string "timeseriesbps" as the second argument.

```

> plot(result.unique, "timeseriesbps")

```

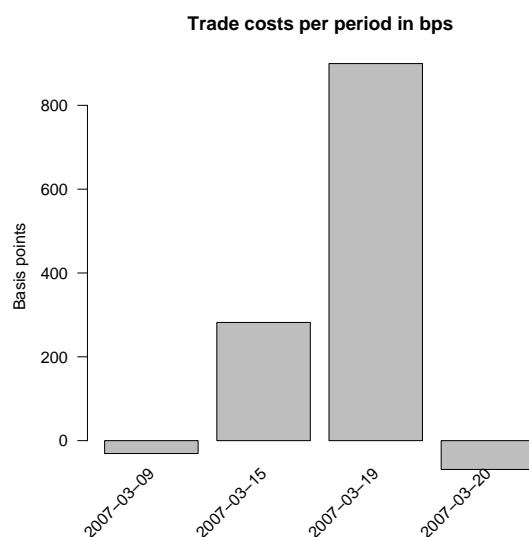


Figure 1: A time series plot of trade costs.

The second bar chart plots trade costs by a column in the static descriptions data frame of class `factor`. For example, if there is a column in the static description data frame called "sector" which lists each of the stocks by their industry, we can specify "sector" as the second argument to the `plot` function.

```
> plot(result, "sector")
```

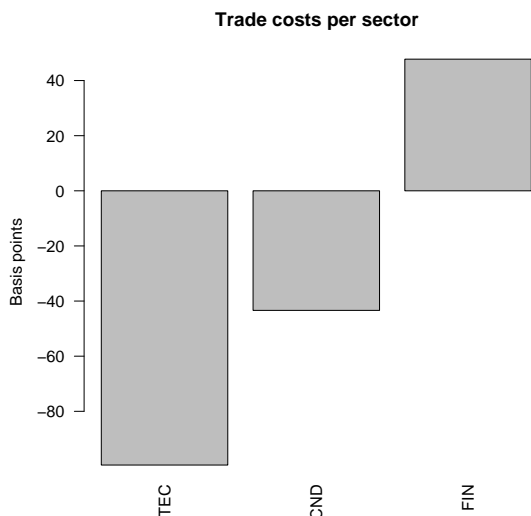


Figure 2: A plot of trade costs per sector.

The last plot we present applies only to same.sided batched trade cost analyses. The batches are grouped by period, and if a batch occurs over multiple periods, it is divided up and these sub-batches are grouped in the appropriate period. Next, the batched trade costs from each period are summed up to produce the following plot:

```
> plot(result.batched, "cumulative")
```

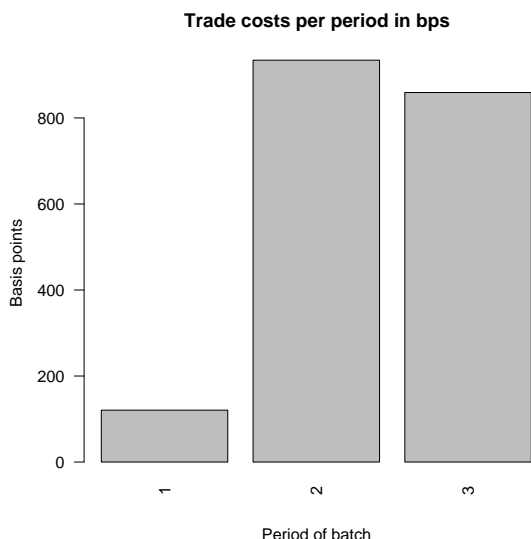


Figure 3: A cumulative plot of trade costs per day, in bps.

## Conclusion

The **tradeCosts** package provides a simple interface to calculate some basic trade costs. These trade costs, however only scratch the surface; there are commissions, fees, spreads, opportunity costs, etc. We aim to implement more and more features in the coming versions of **tradeCosts**.

We hope that the **tradeCosts** package provides useful tools to traders to measure the real cost of executing a trade, and measuring the true performance of any trading strategy.

## Bibliography

R. Kissel and M. Glantz. *Optimal Trading Strategies*. American Management Association, 2003.

Wikipedia. VWAP. *Wikipedia*, 2007. URL <http://en.wikipedia.org/w/index.php?title=VWAP&oldid=123856236>.

Jeff Enos, David Kane, Arjun Ravi Narayan, Aaron Schwartz, Daniel Suo, Luyi Zhao  
 Kane Capital Management  
 Cambridge, Massachusetts, USA  
 jeff@kanecap.com, david@kanecap.com, contact@arjunnarayan.com, Aaron.J.Schwartz@williams.edu, dsuo@fas.harvard.edu and luyizhao@fas.harvard.edu