Paper 11201-2016

Finding National Best Bid and Best Offer – Quote by Quote

Mark Keintz, Wharton Research Data Services

ABSTRACT

U.S. stock exchanges (currently there are 12) are tracked in real time via the Consolidated Trade System (CTS) and the Consolidated Quote System (CQS). CQS contains every updated quote from each of these exchanges, covering some 8,500 stock tickers. It provides the basis by which brokers can honor their fiduciary obligation to investors to execute transactions at the best price, i.e. at the NBBO (National Best Bid or Best Offer). With the advent of electronic exchanges and high frequency trading (timestamps are published to the microsecond), data set size (approaching 1 billion quotes requiring 80 gigabytes of storage for a normal trading day) has become a major operational consideration for market behavior researchers recreating NBBO values.

This presentation demonstrates a straightforward use of hash tables for tracking constantly changing quotes for each ticker/exchange combination to provide the NBBO for each ticker at each time point in the trading day.

INTRODUCTION

While not all stocks are listed in every one of the 12 currently active exchanges[1], most are listed in multiple exchanges. For the trader or broker, this means tracking, for each stock, multiple bids[2] and asks[3] (referred to as "offers" in this presentation). The consolidated quote system (started in 1978) provides a single stream of quotes, in chronological order, containing every change in best bid and offer (BBO) at each exchange for each stock. Of course a trader would want to buy (or sell) at the most advantageous price among all these exchanges – i.e. at the National BBO. A single NBBO can be composed of parts of BBO's from more than one exchange – for example, it may be that two exchanges post the best bid, and a third exchange posts the best offer.

The presentation that follows shows how to track the voluminous sequence of quotes (over 100,000 daily for a typical security) and update the NBBO when applicable. The program logic is applicable to any "auction" context in which multiple buyers and sellers repeatedly issue changing bids and offers for a variety of items.

NBBO FOR A SINGLE STOCK – THE "BEFORE" AND THE "AFTER"

Before dealing with multiple stocks, let's look at the NBBO task for a single stock. For example, consider Table 1, containing 22 IBM quotes at the start of June 10, 2015. The quotes originated from various stock exchanges (B, K, N, P, T, X, Y and Z – all in the "EX" column). Ten of the quotes are highlighted (yellow for bids, blue for offers), indicating those that change the NBBO.

	TIME	EX	SYMBOL	BID	BIDSIZ	OFR	OFRSIZ
1	09:30:00.184	K	IBM	166.05	5	166.86	3
2	09:30:00.184	Y	IBM	159.41	1	166.77	1
3	09:30:00.398	T	IBM	166.09	3	166.77	1
4	09:30:00.409	T	IBM	166.09	3	166.64	3
5	09:30:00.409	T	IBM	166.09	3	166.63	6
6	09:30:00.409	T	IBM	166.10	6	166.63	6
7	09:30:00.411	B	IBM	139.53	1	193.12	1
8	09:30:00.411	X	IBM	139.53	1	193.12	1
9	09:30:00.640	T	IBM	166.10	9	166.63	6

[1] Current CQS documentation identifies 17 exchanges, but our recent quote files have consistently shown only 12, as used in our sample programs.
[2] A "bid" contains the price per share and volume ("bid size") a potential buyer will pay for a given stock.
[3] An "ask" (termed an offer in this paper) contains the price and volume ("offer size") a potential seller will accept for a given stock.

	TIME	EX	SYMBOL	BID	BIDSIZ	OFR	OFRSIZ
10	09:30:00.821	Z	IBM	165.50	1	168.00	1
11	09:30:00.821	Z	IBM	165.50	1	167.20	1
12	09:30:01.006	P	IBM	166.05	1	166.49	1
13	09:30:01.006	P	IBM	166.06	6	166.49	1
14	09:30:01.006	Y	IBM	159.41	1	166.55	1
15	09:30:01.378	N	IBM	166.30	5	166.59	8
16	09:30:01.379	P	IBM	166.05	1	166.49	1
17	09:30:01.379	K	IBM	166.04	4	166.86	3
18	09:30:01.379	Y	IBM	159.41	1	166.77	1
19	09:30:01.379	T	IBM	166.10	9	166.64	3
20	09:30:01.379	T	IBM	166.10	3	166.64	3
21	09:30:01.380	X	IBM	139.53	1	166.87	5
22	09:30:01.380	N	IBM	166.12	8	166.59	8

Table 1: First 22 IBM "Local" BBO Quotes for June 10, 2015[4]

The object is to produce the NBBO results in table 2 below[5] from the quote changes in table 1. Table 2 has ten NBBO records, synchronized with the ten records having shaded elements above – the other twelve quote records have no impact on NBBO. The NBBO records represent a change in bid price (BB) or offer price (BO), or their respective sizes (BBSIZ, BOSIZ). Note record 7 below, which differs from record 6 only in BBSIZ).

	TIME	SYMBOL	BB	BBSIZ	BO	BOSIZ
1	09:30:00.184	IBM	166.05	5	166.86	3
2	09:30:00.184	IBM	166.05	5	166.77	1
3	09:30:00.398	IBM	166.09	3	166.77	2
4	09:30:00.409	IBM	166.09	3	166.64	3
5	09:30:00.409	IBM	166.09	3	166.63	6
6	09:30:00.409	IBM	166.10	6	166.63	6
7	09:30:00.640	IBM	166.10	9	166.63	6
8	09:30:01.006	IBM	166.10	9	166.49	1
9	09:30:01.378	IBM	166.30	5	166.49	1
10	09:30:01.380	IBM	166.12	8	166.49	1

Table 2: IBM NBBO Values Corresponding to Quotes in Table 1

SAS® NBBO CODE FOR A SINGLE STOCK – ARRAYS HAVE GRACE

The obvious logic to generate NBBO values is to keep a running status on each of the 12 exchanges, using four arrays (one each for bids, bid sizes, offers, and offer sizes). The process is as follows:

1. Read a quote.
2. Update the element in each array corresponding to the exchange (variable EX).
3. Find the best bid (BB=max of 12 bids) and best offer (BO=min of 12 offers).

[4] BID and OFR are prices. BIDSIZ and OFRSIZ represent the number of trade units the trader is willing to buy or sell. Usually the unit-of-trade is 100 shares – i.e. BIDSIZ=9 means 900 shares.
[5] BB and BBSIZ represent the national best bid price (i.e. maximum BID) and size (the sum of BIDSIZ over all exchanges at the BB price). BO and BOSIZ are the price and size of the national best offer.

4. Sum up the bid/offer sizes over all exchanges that match the best bid/offer.
5. Finally, if there is a change in best bid or offer then output the NBBO record.

The logic is implemented in this simple program:

SAS Code for Single Stock NBBO	**Notes**

`data nbbo_onesymbol (drop=_:);`
` set quotes;`
` where symbol='IBM';`
	Read a single quote for a specific stock.

` array _bp {12} ;`
` array _bs {12} ;`
` array _op {12} ;`
` array _os {12} ;`
 	
` retain _bp: _bs: _op: _os: ;`
	Four 12-element arrays (12 exchanges), containing:
	Bid and offer prices (_bp and _op)
	Bid and offer sizes (_bs and _os)
	
	RETAIN preserves all values from record to record
	
	Note the trailing colon indicates all variables whose name starts with the indicated stem.

` retain bb bbsiz bo bosiz ;`	Best Bid, Size of Best Bid, Best Offer, Size of Best Offer

` _e=indexc('ABDJKMNPTXYZ',ex);`
 	
` _bp{_e}= bid;`
` _bs{_e}= bidsiz;`
` _op{_e}= ofr;`
` _os{_e}= ofrsiz;`
	Because the exchange identifiers are single letters, the INDEXC function easily assigns unique array elements for each exchange (A=1, B=2, D=3, …, Y=11, Z=12)
	
	Then update the "_E'th" element of each array

` bb=max(of _bp:);    bbsiz=0;`
` bo=min(of _op:);    bosiz=0;`
	Get the best bid (maximum bid), best offer (minimum offer) and initialize their sizes to zero

` do _e= 1 to dim(_bp);`
`  if _bp{_e}=bb then bbsiz=bbsiz+_bs{_e};`
`  if _op{_e}=bo then bosiz=bosiz+_os{_e};`
` end;`
	Update the sizes by looping over the arrays, accumulating sizes for exchanges whose bid or offer matches the best bid/offer

` if lag(bb)^=bb or lag(bbsiz)^=bbsiz`
`  or lag(bo)^=bo or lag(bosiz)^=bosiz;`
`run;`
	Use subsetting IF: Output only if there have been any changes in best bid/offer or their sizes

If the quotes file has multiple stocks, but is sorted by stock, then the program above still works, with very little modification. Simply make these three changes:

1. Substitute
   ```
   by symbol;
   ```
   for the where statement. This provides a dummy variable ("first.symbol") indicating whether the record in hand is the beginning of a new symbol.
2. After the retain statement enter
   ```
   if first.symbol then call missing(bb,bbsiz,bo,bosiz,of _:);
   ```
   which initializes all the retain variables if the record in hand is the start of a new symbol.
3. Add a condition ("**`or first.symbol`**") to the subsetting if statement assuring a new nbbo is output at the start of each symbol
   ```
   if lag(bb)^=bb or lag(bbsiz)^=bbsiz or lag(bo)^=bo or lag(bosiz)^=bosiz
   or first.symbol;
   ```
   This modification is needed only for the extremely unlikely case in which the first nbbo for one symbol entirely matches the last nbbo of the prior symbol.

The program is simple and easy to maintain. However the simplicity is obtained by neglecting some efficiencies, which will be addressed in an expanded document..

INTERLEAVED STOCKS – TIME FOR A HASH UP

If the data file replicates the real stream of quotes, then stock symbols are interleaved at each time value and another solution is required. We effectively have to simultaneously maintain the four arrays above for each stock. Each of these arrays has to (1) be identified by stock, and (2) be retained across observations, just as in the single stock case. A ready solution to this problem is the use of hash tables. The hash object will provide a ready technique for to retrieve and update, on a symbol-by-symbol basis, the running set of BBO's and NBBO's. The program below demonstrates:

SAS Code for Multiple Stock NBBO	**Notes**

`data dummy;`
`  length symbol $6;`
`  retain`
`   symbol  ' '`
`   BB BBSIZ _bp1-_bp12 _bs1-_bs12`
`   BO BOSIZ _op1-_op12 _os1-_os12 .;`
`  stop;`
`run;`
	This is a dummy data set file. It has zero observations (because a **stop** statement precedes the **run** statement) and 53 variables.
	
	This data set will greatly simplify hash construction in the next data step. In particular, it lets the "**h.definedata**" method below use the parameter **all:'YES'** to include all 53 variables brought into the hash object from this data set. Otherwise those 53 variable names would have to be listed.
	
	The 53 variables are all that is needed to trace the status of bids, offers and best bid/offers for each stock symbol.

`data nbbo_many_symbols;`
`  set nbbo.quotes_many_symbols;`
 	
`  if 0 then set dummy;`
`  if _n_=1 then do;`
`   dcl hash h (dataset:'dummy'`
`            ,hashexp:8);`
 	
`     h.definekey('symbol');`
`     h.definedata(all:'YES');`
`     h.definedone();`
`  end;`
	Read a quote record.
	
	The "if 0 then set" statement populates the program data vector with the variables in DUMMY, which is needed for successful use of the following hash object.
	
	By default, hash object contents persist across observations (no "retain" statement necessary). So we "instantiate" it only once. Hence the hash object declare statement ("**dcl**") is inside the "**if _n_=1 then do**" section.
	
	The "**definekey('symbol')**" method tells SAS that items in the hash table are to be "indexed" by symbol, for lookup purposes (i.e. the "find" and "replace" methods below). Each symbol in the hash table will be attached to a unique set of values for the data variables.
	
	The definedata statement tells sas what data variables to include in the hash object – in this case all the variables in dataset dummy.

`  array _bp {12}  ;`
`  array _bs {12}  ;`
`  array _op {12}  ;`
`  array _os {12}  ;`
	These arrays are for bid/offer prices (_bp, _bo) and sizes (_bs, _os). Note the array variables are NOT retained. That is because they will be stored in and retrieved from the hash table, which itself is "retained" by default.

`  _rc=h.find(key:'symbol');`
 	
`  if _rc^=0 then call`
`    missing(bb,bbsiz,bo,bosiz,of _:);`
	Get the hash item (53 variables) for the current symbol.
	
	_RC^=0 means the find method failed – i.e. there is no item in the table corresponding to symbol, so the bid/offer variables need to be reset to missing values.

`_oldbb=bb;`
`_oldbbsiz=bbsiz;`
`_oldbo=bo;`
`_oldbosiz=bosiz;`
	Before finding the new NBBO, store the old values, just now retrieved from the hash table. The old values will be needed to detect any change in the new values.

`_e=indexc('ABDJKMNPTXYZ',ex);`
`_bp{_e}=bid;`
`_bs{_e}=bidsiz;`
`_op{_e}=ofr;`
`_os{_e}=ofrsiz;`
	Update the arrays just as in the single-stock case.

`bb=max(of _bp:);    bbsiz=0;`
`bo=min(of _op:);    bosiz=0;`
 	
`do _e= 1 to dim(_bp);`
`  if _bp{_e}=bb then`
`    bbsiz=bbsiz+_bs{_e};`
`  if _op{_e}=bo then`
`    bosiz=bosiz+_os{_e};`
`end;`
	Update the best bid/offer values, as in the single stock case.

`rc=h.replace(key:'symbol');`	Put updated values for the current stock in the hash object, for later retrieval.

`  if  bb^=_oldbb or bbsiz^=_oldbbsiz`
`   or bo^=_oldbo or bosiz^=_oldbosiz;`
`run;`
	As in the single stock case, output an NBBO only when it is changed.

Unlike array statements, which require a fixed size (12 in this case), there is no similar size specification for the hash object. This demonstrates another advantage of hash objects – dynamic expandability. True, a fixed number of "buckets" is always assigned to a hash table – in the above the "hashexp:8" specifies 2**8=256 buckets. But as more symbols are added, each bucket simply receives more items. In fact SAS documentation[6] states that *"each bucket can hold an infinite number of items"*. At some point, increasing the hashexp value (maximum is 20) might improve performance, but I have not found it to have an impact on the CQS quotes datasets above.

EFFICIENCY CONSIDERATIONS – FUTURE AGENDA

This paper focuses on code simplification, but at the likely expense of significant inefficiencies. In particular it ignores the fact that most incoming quotes have no impact on NBBO – i.e. they are both "inferior" to the current NBBO and do not originate from an exchange issuing the prior best bid/offer (meaning the new quote is not a withdrawal of a previous best bid/offer).. Or the incoming quote may have a superior bid/offer making the need to find the max bid (min offer) superfluous. Code to detect and process these situations (and also quotes that exactly match the current best bid/offer) can be used, avoiding entirely the need for the max (min) functions and subsequent looping through the array to get the best bid/offer size. More complex yes – but also more efficient. I'll discuss just how efficient in the presentation.

CONCLUSION

At first look, generating the national best bid and offer (NBBO) for each of 8,500 stocks from constantly changing quotes might appear to need relatively dense programming, but that need not be the case. With the use of arrays, finding NBBO's for a single stock is very straightforward. For multiple stocks the use of hash tables preserves the

[6] See the "Declare Statement, Hash and Hash Iterator Objects" description in the SAS(R) 9.2 Language Reference Dictionary

programming simplicity. The techniques shown are also readily generalizable to almost any market in which both buyers and sellers frequently change the terms they are prepared to accept.

ACKNOWLEDGMENTS

Many thanks to Rabih Moussawi of Wharton Research Data Services – the motivator and author of the initial single-stock version of the program presented here.

CONTACT INFORMATION

This presentation is part of a work in progress. Your comments and questions are valued and encouraged. Contact the author at:

Name:	Mark Keintz
Enterprise:	Wharton Research Data Services
Address:	3819 Chestnut St, St. Leonards Court Suite 300
City, State ZIP:	Philadelphia, PA 19104
Work Phone:	215/898-2160
E-mail:	mkeintz@wharton.upenn.edu