orderbook

David Kane, Andrew Liu and Khanh Nguyen

Introduction

The **orderbook** package provides facilities for exploring and visualizing the data associated with an order book. An order book keeps track of the outstanding limit orders for a financial instrument, e.g. a stock. A *limit order* is an order to buy or sell a given quantity of stock at a specified limit price or better. The number of shares to be bought or sold is known as the *size*. An order remains in the order book until fully executed, i.e. until its size is zero as a result of trades. Partial executions occur as a result of trades for less than the entire size of the order. In this case the order remains in the order book at the same price, but with an appropriately reduced size.

Consider a simple order book containing five limit orders: sell 150 shares of IBM at $11.11, sell 150 shares of IBM at $11.08, buy 100 shares of IBM at $11.05, buy 200 shares of IBM at $11.05, and buy 200 shares of IBM at $11.01.

```
                Price          Ask Size

                $11.11         150
                $11.08         100
     300        $11.05
     200        $11.01

Bid Size        Price
```

Orders on the *bid* (*ask*) side represent orders to buy (sell). The price levels are $11.11, $11.08, $11.05, and $11.01. The *inside market* is composed of the *best bid* at $11.05 (highest bid price) and the *best ask* at $11.08 (lowest ask price). The *spread* ($0.05) is the difference between the best bid and best ask. The *midpoint* ($11.065) is the average of the best bid and best ask. Every order in an order book has a unique identifier and timestamp indicating the time at which it was accepted into an order book.

There are four types of messages that traders can submit to an order book: *add*, *cancel*, *cancel/replace*, and *market order*. A trader can *add* a limit order into the order book. She can also *cancel* an order and remove it from the order book.

If a trader wants to reduce the size of her order, she can issue a *cancel/replace*, which cancels the order, then immediately replaces it with another order at the same price, but with a lower size.

All orders have a *time priority* depending on when they were accepted into the order book. Earlier orders are executed before later orders. Note that cancel/replace orders can lower the size of an order, but not increase it. Cancel/replace orders maintain the time priority of the original order, so if size increases were allowed, traders with orders at the highest time priority for a price level could perpetually increase the size of their order, preventing others from being able to buy or sell stock at that price level.

A trade occurs when an order is executed through a *market order*, which is an order to immediately buy or sell a given quantity of stock at the best available prices. In the above example, suppose that the order to buy 100 shares at $11.05 was submitted before the order to buy 200 shares at $11.05. The first order has priority, so if a market order to buy 200 shares is submitted, the first order to buy 100 shares will be completely executed, and the second order to buy 200 shares will only be partially executed.

Now suppose that the market order was to sell 400 shares. Then the first price level would be filled, and 100 shares from the next price level would be filled:

```
                Price          Ask Size

                $11.11         150
                $11.08         100
     100        $11.01

Bid Size        Price
```

See Johnson (2010) for more details.

Examples

NVIDIA is a graphics processing unit and chipset developer with ticker symbol NVDA. Consider the order book for NVDA at a leading electronic exchange on June 8, 2010. We create the `orderbook` object by specifying the location of our data file.

```
> library(orderbook)

> file <- system.file("data", "sample.txt",
+     package = "orderbook")
> ob <- orderbook(file = file)
> ob <- read.orders(ob, 10000)
> ob

An object of class orderbook
----------------------------
Current orderbook time:    09:35:02
Message Index:             10,000
Bid Orders:                631
Ask Orders:                1,856
Total Orders:              2,487
```

We read in the first 10,000 messages then `show` the object. The order book time is displayed in 24-hour time, so it is currently 9:35:02 AM. The message index indicates which row in the data file the object has read through. The display also shows that there

are 631 bids and 1,856 asks outstanding, for a total of 2,487 orders. This indicates that many earlier orders have been removed through either cancels or trades.

```
> summary(ob)

Current time is 09:35:02

Ask price levels:   540
Bid price levels:   179
Total price levels: 719
-----------------------------
Ask orders:         1,856
Bid orders:         631
Total orders:       2,487
-----------------------------
Spread:             0.02

Mid point:  11
-----------------------------
Inside market

Best Bid:           11.36
Size:               2,700

Best Ask:           11.38
Size:               400
```

Using `summary` the total order information from `show` is repeated. We see that there are 540 ask and 179 bid price levels, for a total of 719. This indicates that many orders are have been submitted at the same price level. The spread is $0.02, and the midpoint is $11.37. The inside market is composed of 2,700 shares offered at the best bid of $11.36 and 400 shares offered at the best ask of $11.38.

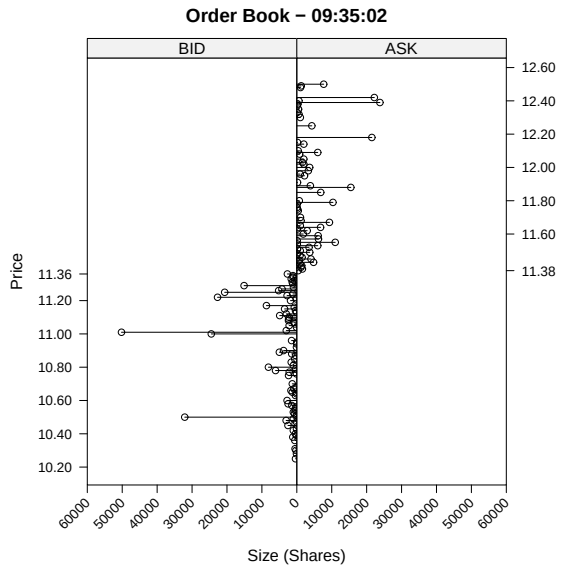
```
> display(ob)

Current time is 09:35:02

                 Price          Ask Size
-------------------------------------------
                 11.42          900
                 11.41          1,400
                 11.40          1,205
                 11.39          1,600
                 11.38          400
-------------------------------------------
  2,700          11.36
  1,100          11.35
  1,100          11.34
  1,600          11.33
    700          11.32
-------------------------------------------
Bid Size          Price
```

`display` shows the inside market, along with the four next best bid and ask price levels and the size at each price level. This gives the user a simple snapshot of the supply and demand in the market.

```
> plot(ob)
```



`plot` is essentially a graphical representation of `display`. Price levels are on the y-axis, and size on the x-axis. The maximum and minimum price levels are 10% above and below the midpoint. Note the large number of shares at $11.01. It is helpful to know whether the depth at that price level is comprised of a single order, or several. Using the `[` method we can view the order information at particular price levels.

```
> ob["11.01"]

  price  size type     time      id
1    11   109  BID 34220988 4403084
2    11 50000  BID 34220988 4403085
3    11   100  BID 34220988 4403086
```

There is an order for 50,000 shares at $11.01 that accounts for almost all of the size. We can view a plot of the number of orders rather than the number of shares at each price level by specifying `type = 'o'` when using `plot`. In the previous plot the maximum and minimum price levels were 10% off from the midpoint, but for this plot we specify a bound of only 3.3% above and below the midpoint (on next page).

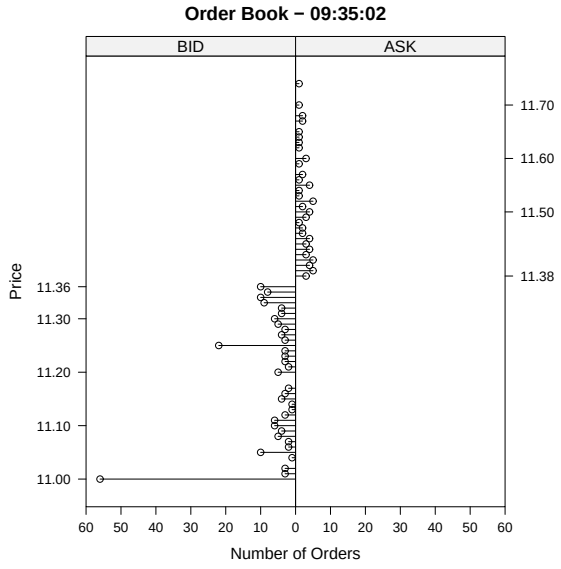
Note the large number of orders at $11.00. The `[` method returns a `data.frame`, so we can use `nrow` to find out the number of orders at $11.00.

```
> nrow(ob["11.00"])

[1] 56
```

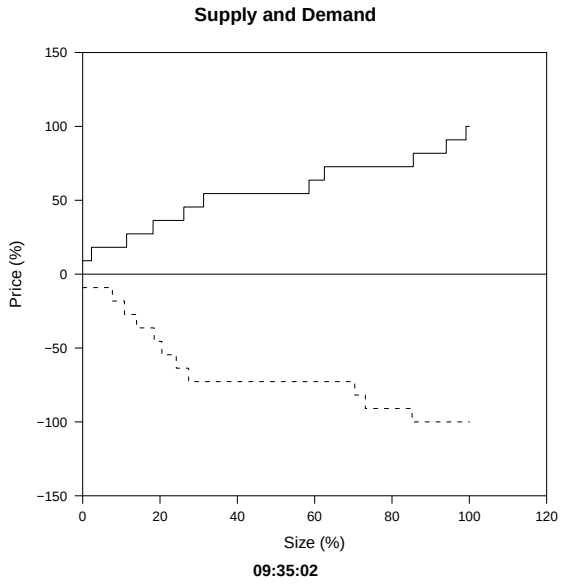
There are 56 orders at that price level, which confirms what we see in the plot.

```
> plot(ob, bounds = 0.033, type = "o")
```



The following plot shows the supply and demand curves for the order book. The demand (supply) curve is downsloping (upsloping). This is because more people want to buy (sell) a stock when the price decreases (increases). The ask (bid) prices are normalized by the absolute value of the difference between the highest (lowest) plotted ask (bid) price level and the the midpoint. The sizes are normalized by the sum of the sizes across all plotted price levels for each side. See Cao et al. (2009) for more detail.

```
> plot(ob, bounds = 0.01, type = "sd")
```

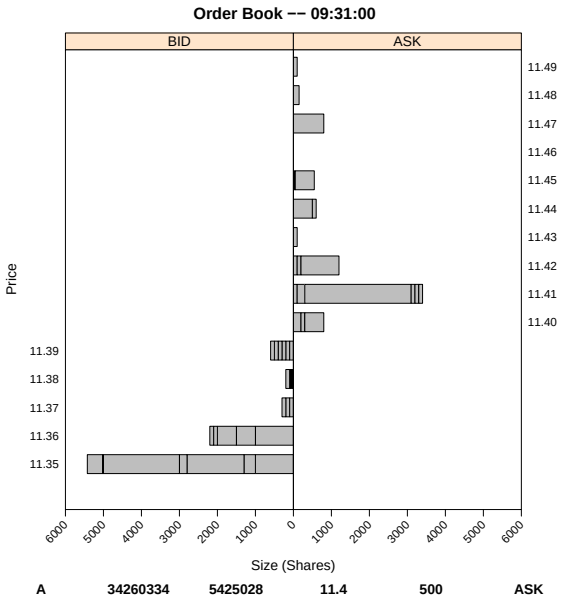



```
> ob <- load.animation(ob, from = "9:30:00",
+     to = "9:31:00", by = "sec")
```

We can also view a simple animation of the order book between two times using the `loadanimation` method. This generates a Trellis object for each second between 9:30:00 and 9:31:00.

```
> ob <- load.animation(ob, from = 500, to = 550,
+     by = "msg")
```

Specifying `by = 'msg'` allows us to see the order book change as a result of individual messages. This generates Trellis objects for each message between message 500 in the data file and message 550. The objects are saved in the R temporary files folder, and their location is stored in the `animation` slot within the `orderbook` object. A single Trellis object generated using `by = 'msg'` is shown below.



Black lines separate individual orders. The order nearest to (furthest from) the middle axis has the highest (lowest) time priority for that price level. `loadanimation` fixes the y- and x-axis labels throughout the animation. The y-axis is chosen so that five price levels above and below the midpoint are visible at all times, while the x-axis is chosen so that it never changes. The current message is printed at the bottom of the plot if `by = 'msg'`. Often it is also useful to watch an animation of the orderbook action around the time of a trade.

```
> ob <- load.trade.animation(ob, before = 30,
+     after = 30, by = "sec")
```

This finds the next trade from the current order book time and generates an animation from 30 seconds before the trade to 30 seconds after the trade. If we specified `by = 'message'` an animation for 30 messages before and after the trade are shown. We can watch these animations using `animate`.

```
> animate(ob, pause = 0.25)
```

The `animate` method contains a loop that prints each object. `pause` specifies the number of seconds to wait in between printing the next Trellis object.

Aside from the ability to retrieve summary statistics and create plots, **orderbook** has methods for creating new `orderbook` objects at different times of interest.

```
> ob <- read.time(ob, "9:30:00")
```

`read.time` returns an `orderbook` object at the first message after the specified time. For example, this returns the `orderbook` object at 9:30:00.

```
> ob <- previous.trade(ob)
> ob

An object of class orderbook
--------------------------
Current orderbook time:    09:34:59
Message Index:             9,958
Bid Orders:                622
Ask Orders:                1,856
Total Orders:              2,478
```

`previous.trade` (`next.trade`) returns an `orderbook` object at the file index of the first trade that occurred before (after) the current order book time. In this case `previous.trade` returns the `orderbook` object at the first trade to occur before 9:30:00.

```
> ob <- read.orders(ob, n = -50)
> ob

An object of class orderbook
--------------------------
Current orderbook time:    09:34:55
Message Index:             9,908
Bid Orders:                616
Ask Orders:                1,860
Total Orders:              2,476
```

`read.orders` is used to move forwards or backwards in the order book by a specified number of messages. In this case, an `orderbook` object at 50 messages before the current message is returned.

Data

Most brokers and exchanges have their own format for transmitting raw order data to customers, so it would be unfeasible for us to write scripts to automatically process all data formats. Consequently, raw data for an `orderbook` object must be in the following form:

```
type,time,id,price,size,type
A,31285893,1231884,11.49,200,ASK
R,31295779,1231884,150
T,31295779,1231884,11.49,50
C,31295781,1231884
```

where A, R, T, C mean Add, Replace, Trade, and Cancel, respectively. The second column is the timestamp of the message in milliseconds after midnight of the users timezone, and the third column is the ID of the order. For a cancel/replace the next column is the new size, while for Add and Trade a column for price comes before the size column. Add messages also have the type of order (BID/ASK) in the sixth column.

In this example an order to sell 200 shares at $11.49 is added to the order book, followed by a cancel/replace and a trade several seconds later. Note that the cancel/replace and the trade have the same timestamp and ID. This is because the `orderbook` object needs to be told the new share size after the trade occurs, in addition to information on the trade. It will not adjust the size of a previous order after a trade occurs without an accompanying cancel/replace present. A trade that leads to a full execution should be accompanied by a cancel message. We see that a few milliseconds after the trade the order is entirely cancelled.

Simulation

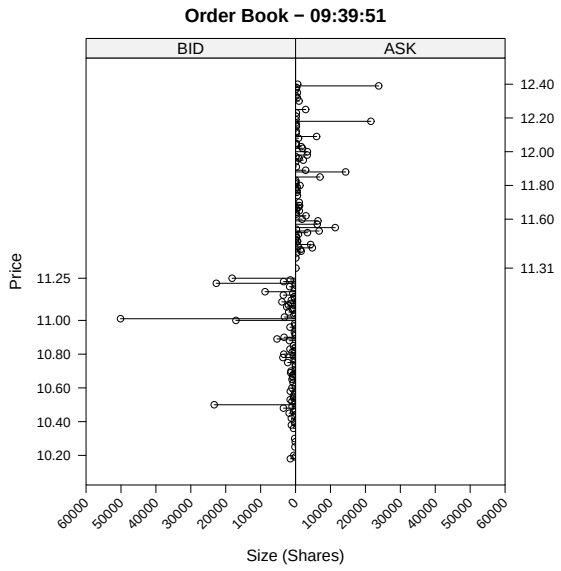
orderbook supports adding, replacing, and cancelling orders. Add orders require the price, size, and type (ASK/BID) of the limit order. Time and ID are optional, and will default to the maximum time + 1 and the maximum ID + 1. Replace messages require the new size and ID. Cancel orders only require ID. In addition, market orders can be issued to the order book. Market orders require size and side (BUY/SELL).

```
> ob <- add.order(ob, 11.2, 300, "ASK")
> ob <- remove.order(ob, 1231883)
> ob <- replace.order(ob, 1231883, 150)
> ob <- market.order(ob, 200, "BUY")
```

Using these tools, the user can write functions to simulate the an order book. In the following example, we consulted Gilles (2006). We simulate 1,000 messages. The messages are chosen based on the following probabilities: 50% for a cancel message, 20% for a market order, and 30% for a limit order. In the event of a cancel message the order cancelled is randomly chosen. Market order have a 50-50 chance for a buy or sell order. The size of the market order always corresponds to the size of the individual order at the best ask or bid with the highest time priority. Limit orders have a 50-50 chance to be an ask or bid. There is a 35% chance for the price of a limit order to be within the spread. If the price is outside of the spread, a price is chosen using a power law distribution. Finally, the size follows a log-normal distribution. A plot of this example simulation is shown below.

```
> ob <- simulate(ob)
```

```
> plot(ob)
```



Conclusion

The current release of the **orderbook** package is meant to serve as a proof-of-concept. Relatively sophisticated order book analytics are possible using an open source package. The **orderbook** package is part of a collection of packages for performing tests of financial conjectures. See Campbell et al. (2007) and Kane and Enos (2006) for more information on the **backtest** and **portfolio** packages, respectively.

David Kane, Andrew Liu and Khanh Nguyen
Kane Capital Management
Cambridge, MA, USA
`dave@kanecap.com`, `Andrew.T.Liu@williams.edu`, *and* `knguyen@cs.umb.edu`

Bibliography

K. Campbell, J. Enos, D. Gerlanc, and D. Kane. Backtests. *R News*, 7(1):36–41, April 2007.

C. Cao, O. Hansch, and X. Wang. The information content of an open limit-order book. *The Journal of Futures Markets*, 29(1):16–41, 2009.

D. Gilles. *Asynchronous Simulations of a Limit Order Book*. PhD thesis, University of Manchester, 2006.

B. Johnson. *Algorithmic Trading & DMA: An introduction to direct access trading strategies.* 4Myeloma Press, London, 2010.

D. Kane and J. Enos. Analysing equity portfolios in R. *R News*, 6(2):13–19, May 2006.