orderbook

David Kane, Andrew Liu and Khanh Nguyen

Introduction

The **orderbook** package provides facilities for exploring and visualizing the orderbook data of financial instruments (stocks, bonds, options, et cetera). An order book keeps track of the outstanding limit orders for a stock. A *limit order* is an order to buy or sell a security at a specified limit price, or better. 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.

The order book would then be:

```
                 Price          Ask Size

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

Bid Size         Price
```

where orders on the *bid* (*ask*) side represent orders to buy (sell), and *size* is the number of shares offered at each price level. The *inside market* is composed of the *best bid* (highest bid price) and *best ask* (lowest ask price) at $11.05 and $11.08, respectively. The *spread* ($0.05) is the difference between the best bid and best ask and the *midpoint* ($11.065) is the average of the best bid and best ask. In addition to price and size, all orders in the order book have a unique identifier and a timestamp representing the time at which they were submitted.

There are four types of messages that market participants can submit to an exchange: *add*, *cancel*, *cancel/replace*, and *market orders*. Participants can either *add* a passive limit order or add a limit order that *sets the market*. In the former a new buy (sell) order is added with a price lower (higher) than the current best bid (ask), while in the latter the buy (sell) order has a price that is higher (lower) than the current best bid (ask). Orders can also be *cancelled* and removed from the order book. If a participants want to update the size of their order, they can issue a *cancel/replace*, which cancels an order, then immediately replaces it with another order at the same price, but with a lower size. In both cases orders are identified by ID.

Note that cancel/replace orders can lower the size of an order, but not increase it. This is because cancel/replace orders maintain the *time priority* of an order. In the case of two or more orders at the same price level, the order accepted first has priority in the event of a *market order*. A market order is an order to immediately buy or sell 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 filled, and the second order to buy 200 shares will only be partially filled.

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
```

There are more nuances in order book microstructure than there is space here to discuss them. Interested readers should refer to Johnson (200) for detailed background.

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 giving the object the location of our data file. Initially it has 0 orders, so we read in the first 10,000 and then `show` the object.

```
> 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
```

The orderbook 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 order book 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 have either been cancelled or removed through trades.

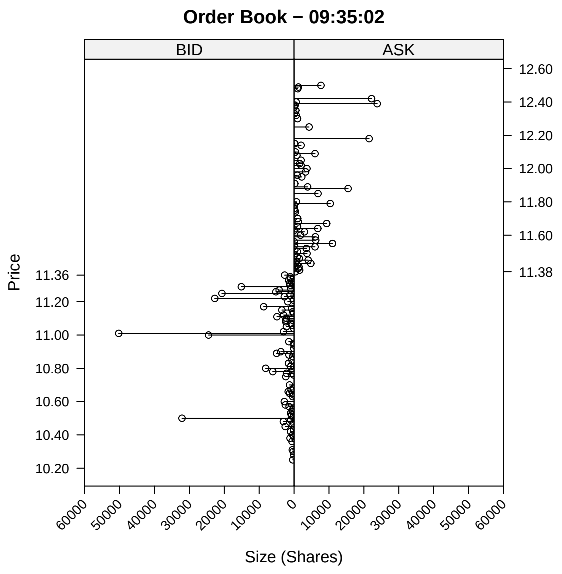
```
> summary(ob)

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:          11.370
-----------------------------
Inside market

Best Bid:           11.36
Size:               2,700

Best Ask:           11.38
Size:               400
```

Using summary the total order information from above is repeated. Additionally, 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 $11.370. 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. This is important because almost all trades occur within the inside market.

```
> 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, along with the size at each price level. This gives the user a simple snapshot of the supply and demand in the market.

```
> plot(ob)
```



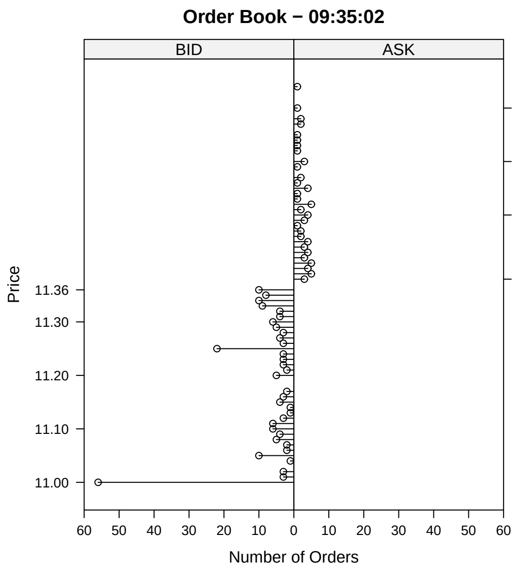
plot shows a graphical representation of the order book with price levels on the y-axis, and size on the x-axis. The maximum and minimum price levels are 10% above and below the midpoint. Note the large order at $11.01. It is helpful to know whether the depth at that price level is comprised of a single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

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

[1] 56

> 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
```

This retrieves the orders at the price levels specified. We see that even though there are 56 orders at $11.00 and only 3 at $11.01, there is an order for 50,000 shares at the latter price level that accounts for almost all of the size.

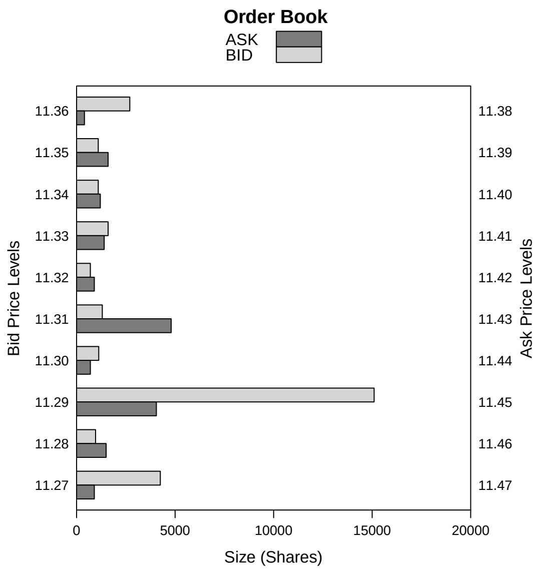
We can view a plot of the orders at each price level by specifying type = 'o' when using plot. In the previous plot the maximum and minimum price levels were 10% off from the midpoint, but for this plot we specify a bound of only 3.3% above and below the midpoint.

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



Viewing the orderbook with bids on one side and asks on another is useful, but users may want to view them side by side to more directly compare the supply and demand at each price level.

```
> plot(ob, type = "s")
```



The user can also view a simple animation of the order book between two times. A frame of the animation is displayed below.

Note that the raw text of the last order book message is displayed at the bottom if individual messages are being read (discussed further below). The black lines separate orders, with the orders nearest the middle y-axis having the highest time priority. The `loadanimation` method will show price levels at least five pennies above and below the best ask and best bid, as well as the pennies in between for each second between 9:30:00 and 9:31:00. To ensure that the axes do not change the method first creates all order books between the start and stop time and finds the maximum size, as well as the maximum and minimum price levels to be plotted. These values are used to set the y- and x-axis. Then Trellis objects are created for each order book and stored in a file. The location of the file is stored in a slot in the `orderbook` object.

```
> ob <- loadanimation(ob, "9:30:00", "9:31:00")
```

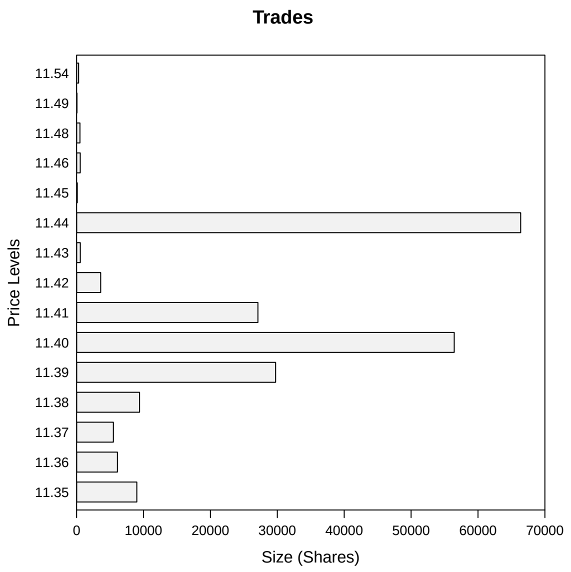
If the users want to “slow down” the order book near the end time, they can add a slow parameter that has the animation generate order books for each individual message, instead of each second. The following will create order books for each second between the start time and the time 50 messages before 9:31:00, then it will create orderbooks for each message from that point until 9:31:00.

```
> ob <- loadanimation(ob, "9:30:00", "9:31:00",
+     slow = 50)
```

`animate` is a simple loop that prints the objects. `pause` specifies the number of seconds to wait in between printing the next Trellis object.

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

```
> plotTrade(ob)
```



Finally, users can plot the trade data by using `plot.trade`. This creates a simple bar graph of the number of shares traded at each price level.

Aside from the ability to retrieve summary statistics and create graphics, **orderbook** can create different `orderbook` objects for viewing the order book at different times. For example, the user may want to view the order book when the market opens at 9:30:00.

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

Suppose the user wants to view the last pre-market trade. `previous.trade` finds the first trade that occurred before the current order book time, and then returns an `orderbook` object at that time. `next.trade` finds the next trade that occured after the current order book time, and then returns an `orderbook` object at that time.

```
> 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
```

`read.orders` is used to move forwards or backwards in the order book by a specified number of messages. The following will change the state of the orderbook to 50 messages previous to the current message.

```
> 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
```

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 that data. 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 first column is the timestamp of the message in milliseconds after midnight of the users timezone, and the second column is the ID of the order. For a cancel/replace the next number is the new size, while for Add and Trade price comes before size, followed by the type of order in the case of Add (BID/ASK).

In this example an order to sell 200 shares at $11.49 is added to the orderbook, 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` needs to be told the new share size after the trade occurs, as well as information on the trade. It will not adjust the size of a previous order after a trade occurs without an accompanying cancel/replace present. We see that a few milliseconds after the trade the order is entirely cancelled.

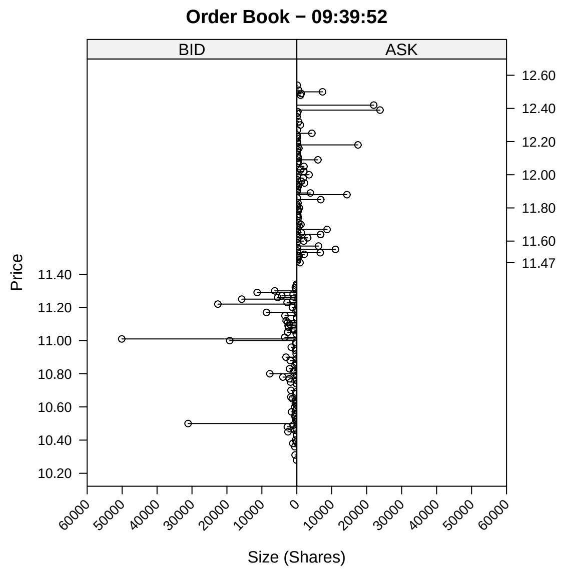
Simulation

orderbook supports adding, replacing, and cancelling orders. To add an order, the user needs to specify the price, size, and type. Time and ID are optional, and will default to the maximum time and the maximum ID + 1, respectively. For replacing an order, only ID and size need to be given, and for cancelling an order, only ID is necessary. Market orders are also possible by specifying the size and side (BUY/SELL).

```
> display(ob)
> ob <- add.order(ob, stuff)
> ob <- remove.order(ob, stuff)
> ob <- replace.order(ob, stuff)
> ob <- market.order(ob, 200, "BUY")
> display(ob)
```

Using these tools, the user can write functions to simulate the movement of an order book. In the following example, we consulted Gilles (2006). We simulate 1,000 orders. In each iteration of our simulation there is a 50% chance for a cancel order to be placed, 20% chance for a market order, and 30% chance for a limit order. Orders are cancelled completely randomly, and for a market order there is a 50-50 chance for a buy or sell order to be placed. The size of the market order always corresponds to the size of the best ask or bid at the front of the queue. When a limit order is placed, there is a 50-50 chance for it to be an ask or bid. Then there is a 35% chance for the price to be within the spread, in which case a price is chosen based on a uniform distribution. If the price is determined to be outside of the spread, a price is chosen using a power law distribution. The size follows a log-normal distribution.

```
> 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. 2007.

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, 200.

D. Kane and J. Enos. Analysing equity portfolios in r. *R News*, 6(2):13–19, May 2006. URL `http://CRAN.R-project.org/doc/Rnews`.