Package: obAnalytics (via r-universe)

September 9, 2024

Title Limit Order Book Analytics

Version 0.1.2

Description Data processing, visualisation and analysis of Limit Order Book event data.

Date 2018-10-09

URL `https://github.com/phil8192/ob-analytics`

BugReports `https://github.com/phil8192/ob-analytics/issues`

License GPL (>= 2)

Depends R (>= 3.1.1)

Imports zoo, ggplot2, reshape2, utils

Suggests grid, rmarkdown, knitr, testthat

LazyData true

VignetteBuilder knitr

RoxygenNote 6.1.0

Encoding UTF-8

Repository https://phil8192.r-universe.dev

RemoteUrl https://github.com/phil8192/ob-analytics

RemoteRef HEAD

RemoteSha 97d1c08c10ac90411d329d08254f23dba02080d8

Contents

`obAnalytics-package` *obAnalytics.*

Description

Limit order book analytics.

Main functionality

- Limit order book event processing.
- Visualise order book state and market impacts.
- Order book reconstruction and analysis.

Data processing

The main focus of this package is reconstruction of a limit order book. The `processData` function will perform data processing based on a supplied CSV file, the schema of which is defined in the `processData` function documentation. Example preprocessed limit order data are also provided (see `lob.data`) which has been derived from the example raw data provided the inst/extdata directory.

The data processing consists of a number of stages:

- Cleaning of duplicate and erroneous data.
- Identification of sequential event relationships.
- Inference of trade events via order-matching.
- Inference of order types (limit vs market).
- Construction of volume by price level series.
- Construction of order book summary statistics.

Limit order events are related to one another by *volume deltas* (the change in volume for a limit order). To simulate a matching-engine, and thus determine directional trade data, volume deltas from both sides of the limit order book are ordered by time, yielding a sequence alignment problem, to which the the Needleman-Wunsch algorithm has been applied.

Visualisation

The package provides a number of functions for the visualisation of limit order events and order book liquidity. The visualisations all make use of the ggplot2 plotting system:

plotTimeSeries General time series plotting.

plotTrades Plot `trades` data.

plotCurrentDepth Visualise the *shape* of an `orderBook`.

plotPriceLevels Visualise volume by price level through time.

plotVolumePercentiles Visualise order book liquidity through time.

plotEventMap Visualise sequential limit order events by price level.

plotVolumeMap Visualise sequential limit order events by volume.

plotEventsHistogram Convenience function.

The `plotPriceLevels` visualisation is designed to show the *ebb and flow* of limit order volume at all price levels including the interplay between the bid/ask spread. It is possible to identify interesting market participant behaviour and to visualise market shocks and resilience with this function.

The `plotEventMap` function is useful for studying systematic market participant behaviour. Interesting sequential patterns can be observed in this visualisation as algorithms react to various market events by repositioning orders.

The `plotVolumeMap` function shows a visualisation of cancelled volume through time. It is possible to identify and filter out individual trading algorithms from this graph.

The `plotVolumePercentiles` visualisation is inspired by the size map chart included in many articles from Nanex research and is intended to show available market liquidity.

In all visualisations it is possible to filter the data by time, price and volume.

Analysis

In addition to the generated `lob.data` which are intended to be used as a basis for further research, the package currently provides a limited set of trade and order book analysis functions:

filterDepth Filter `depth` data by time period.

getSpread Extract the bid/ask quotes from the `depth.summary` data.

orderBook Reconstruct a Limit order book from `events` data.

tradeImpacts Group `trades` into individual impact events.

Additional functionality will be added to the package in the future.

Author(s)

Philip Stubbings <`phil@parasec.net`>

References

`http://parasec.net/transmission/order-book-visualisation`

`depth` *Depth.*

Description

Price level depth (liquidity) through time.

Format

A data.frame consisting of the following fields:

timestamp Time at which volume was added or removed.

price Order book price level.

volume Amount of remaining volume at this price level.

side The side of the price level: *bid* or *ask*.

Details

The depth data.frame describes the amount of available volume for all price levels in the limit order book through time. Each row corresponds to a limit order event, in which volume has been added or removed.

Author(s)

phil

See Also

Other Limit order book data: `depth.summary`, `events`, `trades`

`depth.summary` *Depth summary.*

Description

Limit order book summary statistics.

Format

A data.frame consisting of the following fields:

timestamp Local timestamp corresponding to `events`.

best.bid.price Best bid price.

best.bid.vol Amount of volume available at the best bid.

bid.vol25:500bps The amount of volume available for 20 25bps percentiles below the best bid.

best.ask.price The best ask price.

best.ask.vol Amount of volume available at the best ask.

ask.vol25:500bps The amount od volume available for 20 25bps percentiles above the best ask.

Details

Various summary statistics describing the state of the order book after every limit order event. The metrics are intended to quantify the *shape* of the order book through time.

Author(s)

phil

See Also

Other Limit order book data: `depth`, `events`, `trades`

`events` *Limit order events.*

Description

A data.frame containing the lifecycle of limit orders.

Format

A data.frame consisting of the following fields:

event.id Event ID.

id Limit Order ID.

timestamp Local timestamp for order update (create/modify/delete).

exchange.timestamp Exchange order creation time.

price Limit order price level.

volume Remaining limit order volume.

action Event action: created, changed, deleted.

direction Order book side: bid, ask.

fill For changed or deleted events, indicates the change in volume.

matching.event Matching `event.id` if this event is part of a trade. `NA` otherwise.

type Limit order type (see *Event types* below.)

aggressiveness.bps The distance of the order from the edge of the book in Basis Points (BPS).

Each limit order *type* has been categorised as follows:

unknown It was not possible to infer the order type given the available data.

flashed-limit Order was created then subsequently deleted. 96% of example data.

resting-limit Order was created and left in order book indefinitely until filled.

market-limit Order was partially filled before landing in the order book at it's limit price.

market Order was completely filled and did not come to rest in the order book.

pacman A limit-price modified *in situ* (exchange algorithmic order).

Details

The purpose of this table is to keep account of the lifecycle of all orders in both sides of the limit order book. The lifecycle of an individual limit order follows a sequence of events:

created The order is created with a specified amount of volume and a limit price.

changed The order has been partially filled. On each modification, the remaining volume will decrease.

deleted The order may be deleted at the request of the trader or, in the event that the order has been completely filled, deleted by the exchange. An order deleted by the exchange as a result of being filled will have 0 remaining volume at time of deletion.

Author(s)

phil

See Also

Other Limit order book data: `depth.summary`, `depth`, `trades` `filterDepth` *Filter price level volume.*

Description

Given depth data calculated by `priceLevelVolume`, filter between a specified time range. The resulting data will contain price level volume which is active only within the specified time range.

Usage

```
filterDepth(d, from, to)
```

Arguments

`d`	`depth` data.
`from`	Beginning of range.
`to`	End of range.

Details

For price levels with volume > 0 before the time range starts, timestamps will be set to the supplied `from` parameter.

For volume > 0 after the time range ends, timestamps will be set to the supplied `to` parameter and volume set to 0.

For example, the following data taken from `priceLevelVolume` for price level 243.29 shows the available volume through time at that price level between `00:52:37.686` and `03:28:49.621`.

timestamp	price	volume	side
2015-05-01 00:52:37.686	243.29	911500000	ask
2015-05-01 01:00:36.243	243.29	862200000	ask
2015-05-01 02:45:43.052	243.29	0	ask
2015-05-01 02:52:24.063	243.29	614700000	ask
2015-05-01 02:52:51.413	243.29	0	ask
2015-05-01 02:53:13.904	243.29	952300000	ask
2015-05-01 03:28:49.621	243.29	0	ask

applying `filterDepth` to this data for a time range beteen `02:45` and `03:00` will result in the following:

timestamp	price	volume	side
2015-05-01 02:45:00.000	243.29	862200000	ask
2015-05-01 02:45:43.052	243.29	0	ask
2015-05-01 02:52:24.063	243.29	614700000	ask
2015-05-01 02:52:51.413	243.29	0	ask
2015-05-01 02:53:13.904	243.29	952300000	ask

2015-05-01 03:00:00.000	243.29	0	ask

Note that the timestamps at the begining and end of the table have been *clamped* to the specified range and the volume set to 0 at the end.

Value

Filtered depth data.

Author(s)

phil

Examples

```
# obtain price level volume for a 15 minute window.
filtered <- with(lob.data, filterDepth(depth,
    from=as.POSIXct("2015-05-01 02:45:00.000", tz="UTC"),
    to=as.POSIXct("2015-05-01 03:00:00.000", tz="UTC")))

# top 5 most active price levels during this 15 minute window.
head(sort(tapply(filtered$volume, filtered$price, length),
    decreasing=TRUE), 5)

# extract available volume for price level 233.78, then plot it.
level.233.78 <- filtered[filtered$price == 233.78, c("timestamp", "volume")]
plotTimeSeries(level.233.78$timestamp, level.233.78$volume*10^-8)
```

`getSpread` *Get the spread.*

Description

Extracts the spread from the depth summary, removing any points in which a change to bid/ask price/volume did not occur.

Usage

```
getSpread(depth.summary)
```

Arguments

`depth.summary`	`depth.summary` data.

Details

The spread (best bid and ask price) will change following a market order or upon the addition/cancellation of a limit order at, or within, the range of the current best bid/ask. A change to the spread that is *not* the result of a market order (an impact/market shock) is known as a *quote*.

The following table shows a market spread betwen `05:03:22.546` and `05:04:42.957`. During this time, the best ask price and volume changes whilst the best bid price and volume remains static.

timestamp	bid.price	bid.vol	ask.price	ask.vol
05:03:22.546	235.45	16235931	235.72	39375160
05:03:24.990	235.45	16235931	235.72	21211607
05:03:25.450	235.45	16235931	235.71	39375160
05:04:15.477	235.45	16235931	235.72	39058160
05:04:16.670	235.45	16235931	235.71	39058160
05:04:42.957	235.45	16235931	235.71	77019160

Value

Bid/Ask spread quote data.

Author(s)

phil

Examples

```
# get the last 25 quotes (changes to the spread).
with(lob.data, tail(getSpread(depth.summary), 25))
```

`loadData` *Load pre-processed data.*

Description

Loads previously saved pre-processed data.

Usage

```
loadData(bin.file, ...)
```

Arguments

`bin.file`	File location.
`...`	`readRDS`.

Details

Convenience function.

Value

Limit order, trade and depth data structure `lob.data`.

Author(s)

phil

Examples

```
## Not run:

lob.data <- loadData(bin.file="/tmp/lob.data.rds")

## End(Not run)
```

`lob.data` *Example limit order book data.*

Description

50,393 limit order events. 482 trades.

Usage

```
data(lob.data)
```

Format

A list containing 4 data frames as returned by `processData`

Details

5 hours of limit order book event data obtained from the Bitstamp (bitcoin) exchange on 2015-05-01 (midnight until 5am). The data has been preprocessed with the `processData` function.

Author(s)

phil

Source

`https://www.bitstamp.net/websocket`

References

`https://github.com/phil8192/ticker`

See Also

`events`, `trades`, `depth`, `depth.summary`

orderBook *Instantaneous limit order book reconstruction.*

Description

Given a set of `events`, reconstructs a limit order book for a specific point in time.

Usage

```
orderBook(events, tp = as.POSIXlt(Sys.time(), tz = "UTC"),
  max.levels = NULL, bps.range = 0, min.bid = 0, max.ask = Inf)
```

Arguments

`events`	Limit order `events` data.frame.
`tp`	Time point to re-construct order book at.
`max.levels`	Max number of price levels to return.
`bps.range`	Max depth to return +- BPS from best bid/ask.
`min.bid`	Min bid to return.
`max.ask`	Max ask to return.

Details

An order book consists of 2 sides: *bids* and *asks*, an example of which is shown below:

id	price	volume	liquidity	bps
65613703	236.58	910229141	6341547077	2.11
65613655	236.56	1320000000	5431317936	1.26
65613700	236.55	1320000000	4111317936	0.84
65613698	236.54	1600000000	2791317936	0.42
65613712	236.53	1191317936	1191317936	0.00
-	-	-	-	-
65613225	236.36	16154172	16154172	0.00
65613681	236.31	200000000	216154172	2.11
65613220	236.30	100000000	316154172	2.53
65612978	236.28	100000000	416154172	3.38
65612388	236.17	100000000	516154172	8.03

Value

Limit Order Book structure. A list containing 3 fields:

timestamp Timestamp the order book was reconstructed for.

asks A data.frame containing the Ask side of the order book.

bids A data.frame containing the Bid side of the order book.

The *bids* and *asks* data consists of the following:

id Limit order Id.

timestamp Last modification time to limit order.

exchange.timestamp Time at which order was placed in order book.

price Limit order price.

volume Limit orer volume.

liquidity Cumulative sum of volume from best bid/ask up until price.

bps Distance (in BPS) of order from best bid/ask.

Both the *bids* and *asks* data are ordered by descending price.

Author(s)

phil

Examples

```
tp <- as.POSIXct("2015-05-01 04:25:15.342", tz="UTC")
orderBook(lob.data$events, max.levels=5)
```

`plotCurrentDepth` *Visualise order book depth at any given point in time.*

Description

Plots the cumalative volume on each side of the limit order book.

Usage

```
plotCurrentDepth(order.book, volume.scale = 1, show.quantiles = T,
  show.volume = T)
```

Arguments

`order.book`	A limit `orderBook` structure.
`volume.scale`	Volume scale factor.
`show.quantiles`	If true, highlight top 1% highest volume.
`show.volume`	If true, also show non-cumulative volume.

Author(s)

phil

Examples

```
# get a limit order book for a specific point in time, limited to +- 150bps
# above/below best bid/ask price.
lob <- orderBook(lob.data$events,
    tp=as.POSIXct("2015-05-01 04:38:17.429", tz="UTC"), bps.range=150)

# visualise the order book liquidity.
plotCurrentDepth(lob, volume.scale=10^-8)
```

---

plotEventMap *Plot limit order event map.*

Description

Generates a visualisation of limit order events (excluding market and market limit orders).

Usage

```
plotEventMap(events, start.time = min(events$timestamp),
  end.time = max(events$timestamp), price.from = NULL,
  price.to = NULL, volume.from = NULL, volume.to = NULL,
  volume.scale = 1)
```

Arguments

`events`	Limit order `events` data.frame.
`start.time`	Plot events from this time onward.
`end.time`	Plot events up until this time.
`price.from`	Plot events with price levels >= this value.
`price.to`	Plot events with price levels <= this value.
`volume.from`	Plot events with volume >= this value relevant to volume.scale
`volume.to`	Plot events with volume <= this value relevant to volume scale.
`volume.scale`	Volume scale factor.

Details

- Ask side orders = red.
- Bid side orders = blue.
- Volume of order determines size of circle.
- Opaque = volume was added.
- Transparent = volume was removed.

Author(s)

phil

Examples

```
## Not run:

# plot all orders
with(lob.data, plotEventMap(events))

## End(Not run)

# 1 hour of activity and re-scale the volume
with(lob.data, plotEventMap(events,
    start.time=as.POSIXct("2015-05-01 03:30:00.000", tz="UTC"),
    end.time=as.POSIXct("2015-05-01 04:00:00.000", tz="UTC"),
    volume.scale=10^-8))

# 15 minutes of activity >= 5 (re-scaled) volume within price range
# $ [220, 245]
with(lob.data, plotEventMap(events,
    start.time=as.POSIXct("2015-05-01 03:30:00.000", tz="UTC"),
    end.time=as.POSIXct("2015-05-01 03:45:00.000", tz="UTC"),
    price.from=220,
    price.to=245,
    volume.from=5,
    volume.scale=10^-8))
```

`plotEventsHistogram` *Plot a histogram given event data.*

Description

Convenience function for plotting event price and volume histograms. Will plot ask/bid bars side by side.

Usage

```
plotEventsHistogram(events, start.time = min(events$timestamp),
  end.time = max(events$timestamp), val = "volume", bw = NULL)
```

Arguments

`events`	Limit order `events` data.
`start.time`	Include event data >= this time.
`end.time`	Include event data <= this time.
`val`	"volume" or "price".
`bw`	Bar width (for price, 0.5 = 50 cent buckets.)

Author(s)

phil

Examples

```
# necessary columns from event data.
events <- lob.data$events[, c("timestamp", "direction", "price", "volume")]

# re-scale volume (if needed)
events$volume <- events$volume * 10^-8

# histogram of all volume aggregated into 5 unit buckets.
plotEventsHistogram(events[events$volume < 50, ], val="volume", bw=5)

# histogram of 99% of limit prices during a 1 hour time frame.
# bar width set to 0.25: counts are aggregated into 25 cent buckets.
plotEventsHistogram(events[events$price <= quantile(events$price, 0.99)
                    & events$price >= quantile(events$price, 0.01), ],
    start.time=as.POSIXct("2015-05-01 02:15:00.000", tz="UTC"),
    end.time=as.POSIXct("2015-05-01 03:15:00.000", tz="UTC"),
    val="price", bw=0.25)
```

---

plotPriceLevels *Plot order book price level heat map.*

Description

Produces a visualisation of the limit order book depth through time.

Usage

```
plotPriceLevels(depth, spread = NULL, trades = NULL, show.mp = T,
  show.all.depth = F, col.bias = 0.1,
  start.time = head(depth$timestamp, 1),
  end.time = tail(depth$timestamp, 1), price.from = NULL,
  price.to = NULL, volume.from = NULL, volume.to = NULL,
  volume.scale = 1, price.by = NULL)
```

Arguments

`depth`	The order book `depth`.
`spread`	Spread to overlay obtained from `getSpread`.
`trades`	`trades` data.
`show.mp`	If True, spread will be summarised as midprice.
`show.all.depth`	If True, show resting (and never hit) limit orders.

`col.bias`	1 = uniform colour spectrum. 0.25 = bias toward 0.25 (more red less blue). <= 0 enables logarithmic scaling.
`start.time`	Plot depth from this time onward.
`end.time`	Plot depth up until this time.
`price.from`	Plot depth with price levels >= this value.
`price.to`	Plot depth with price levels <= this value.
`volume.from`	Plot depth with volume >= this value relevant to volume.scale
`volume.to`	Plot depth with volume <= this value relevant to volume scale.
`volume.scale`	Volume scale factor.
`price.by`	The increment for the 'limit price' scale (y)

Details

The available volume at each price level is colour coded according to the range of volume at all price levels. The colour coding follows the visible spectrum, such that larger amounts of volume appear "hotter" than smaller amounts, where cold = blue, hot = red.

Since the distribution of limit order size exponentially decays, it can be difficult to visually differentiate: most values will appear to be blue. The function provides price, volume and a colour bias range to overcome this.

Author(s)

phil

Examples

```
# bid/ask spread.
spread <- with(lob.data, getSpread(depth.summary))

## Not run:

# plot all depth levels, rescaling the volume by 10^-8.
# produce 2 plots side-by-side: second plot contains depth levels with > 50
# units of volume.
p1 <- with(lob.data, plotPriceLevels(depth, spread,
                                       col.bias=0.1,
                                       volume.scale=10^-8))
p2 <- with(lob.data, plotPriceLevels(depth, spread,
                                       col.bias=0.1,
                                       volume.scale=10^-8,
                                       volume.from=50))
library(grid)
pushViewport(viewport(layout=grid.layout(1, 2)))
print(p1, vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(p2, vp=viewport(layout.pos.row=1, layout.pos.col=2))

## End(Not run)
```

```
# zoom into 1 hour of activity, show the spread and directional trades.
with(lob.data, plotPriceLevels(depth, spread, trades,
   start.time=as.POSIXct("2015-05-01 03:25:00.000", tz="UTC"),
   end.time=as.POSIXct("2015-05-01 04:25:00.000", tz="UTC"),
   volume.scale=10^-8))

# zoom in to 15 minutes of activity, show the bid/ask midprice.
with(lob.data, plotPriceLevels(depth, spread,
   show.mp=FALSE,
   start.time=as.POSIXct("2015-05-01 03:30:00.000", tz="UTC"),
   end.time=as.POSIXct("2015-05-01 03:45:00.000", tz="UTC")))
```

plotTimeSeries *General purpose time series plot.*

Description

Convenience function for plotting time series.

Usage

```
plotTimeSeries(timestamp, series, start.time = min(timestamp),
  end.time = max(timestamp), title = "time series",
  y.label = "series")
```

Arguments

timestamp	POSIXct timestamps.
series	The time series.
start.time	Plot from this time onward.
end.time	Plot up until this time.
title	Plot title.
y.label	Y axis label of the plot.

Author(s)

phil

Examples

```
# plot trades.
with(lob.data$trades, plotTimeSeries(timestamp, price))

# plot a general time series.
timestamp <- seq(as.POSIXct("2015-05-01 00:00:00.000", tz="UTC"),
                 as.POSIXct("2015-05-01 00:59:00.000", tz="UTC"), by=60)
series <- rep(1:10, 6)
```

```
plotTimeSeries(timestamp, series)
```

`plotTrades` *plotTrades.*

Description

A convenience function for plotting the trades data.frame in a nice way.

Usage

```
plotTrades(trades, start.time = min(trades$timestamp),
  end.time = max(trades$timestamp))
```

Arguments

`trades`	`trades` data.
`start.time`	Plot from.
`end.time`	Plot to.

Author(s)

phil

Examples

```
with(lob.data, plotTrades(trades))
```

`plotVolumeMap` *Visualise flashed-limit order volume.*

Description

Plots the points at which volume was added or removed from the limit order book.

Usage

```
plotVolumeMap(events, action = "deleted", type = c("flashed-limit"),
  start.time = min(events$timestamp), end.time = max(events$timestamp),
  price.from = NULL, price.to = NULL, volume.from = NULL,
  volume.to = NULL, volume.scale = 1, log.scale = F)
```

Arguments

`events`	Limit order `events` data.frame.
`action`	"deleted" for cancelled volume, "added" for added volume.
`type`	default = c("flashed-limit"). Set of types.
`start.time`	Plot events from this time onward.
`end.time`	Plot events up until this time.
`price.from`	Plot events with price levels >= this value.
`price.to`	Plot events with price levels <= this value.
`volume.from`	Plot events with volume >= this value relevant to volume.scale
`volume.to`	Plot events with volume <= this value relevant to volume scale.
`volume.scale`	Volume scale factor.
`log.scale`	If true, plot volume on logarithmic scale.

Details

A flashed limit-order is a "fleeting" limit order: an order was added, then removed (usually within a very short period of time). This plot is especially useful for identifying individual trading algorithms by price and volume.

Author(s)

phil

Examples

```
# plot all fleeting limit order volume using logarithmic scale.
with(lob.data, plotVolumeMap(events, volume.scale=10^-8, log.scale=TRUE))

# "fleeting" order volume within 1 hour range up until 10 units of volume.
with(lob.data, plotVolumeMap(events, volume.scale=10^-8,
    start.time=as.POSIXct("2015-05-01 02:30:00.000", tz="UTC"),
    end.time=as.POSIXct("2015-05-01 03:30:00.000", tz="UTC"),
    volume.to=10))
```

`plotVolumePercentiles` *Visualise available limit order book liquidity through time.*

Description

Plots the available volume in 25bps increments on each side of the order book in the form of a stacked area graph.

Usage

```
plotVolumePercentiles(depth.summary,
  start.time = head(depth.summary$timestamp, 1),
  end.time = tail(depth.summary$timestamp, 1), volume.scale = 1,
  perc.line = T, side.line = T)
```

Arguments

`depth.summary`	`depth.summary` data.
`start.time`	Plot events from this time onward.
`end.time`	Plot events up until this time.
`volume.scale`	Volume scale factor.
`perc.line`	If true, separate percentiles with subtle line.
`side.line`	If true, separate bid/ask side with subtle line.

Details

The top of the graph depicts the ask side of the book, whilst the bottom depicts the bid side. Percentiles and order book sides can be separated by an optional subtle line for improved legibility.

Author(s)

phil

Examples

```
# visualise 2 hours of order book liquidity.
# data will be aggregated to minute-by-minute resolution.
plotVolumePercentiles(lob.data$depth.summary,
    start.time=as.POSIXct("2015-05-01 02:30:00.000", tz="UTC"),
    end.time=as.POSIXct("2015-05-01 04:30:00.000", tz="UTC"),
    volume.scale=10^-8)

## Not run:

# visualise 15 minutes of order book liquidity.
# data will be aggregated to second-by-second resolution.
plotVolumePercentiles(lob.data$depth.summary,
    start.time=as.POSIXct("2015-05-01 04:30:00.000", tz="UTC"),
    end.time=as.POSIXct("2015-05-01 04:35:00.000", tz="UTC"),
    volume.scale=10^-8)

## End(Not run)
```

`processData` *Import CSV file.*

Description

Imports and performs preprocessing of limit order data contained in a CSV.

Usage

```
processData(csv.file, price.digits = 2, volume.digits = 8)
```

Arguments

`csv.file`	Location of CSV file to import
`price.digits`	an integer indicating the number of decimal places in 'price' column of the CSV file
`volume.digits`	an integer indicating the number of decimal places in 'volume' column of the CSV file

Details

The CSV file is expected to contain 7 columns:

id Numeric limit order unique identifier

timestamp Time in milliseconds when event received locally

exchange.timestamp Time in milliseconds when order first created on the exchange

price Price level of order event. It will be rounded by round(price, price.digits)

volume Remaining order volume. It will be rounded by round(price, volume.digits)

action Event type (see below)

direction Side of order book (bid or ask)

action describes the limit order life-cycle:

created The limit order has been created

modified The limit order has been modified (partial fill)

deleted The limit order was deleted. If the remaining volume is 0, the order has been filled.

An example dataset returned from this function can be seen in `lob.data` which is the result of processing the example data included in the `inst/extdata` directory of this package.

Value

A list containing 4 data frames:

events Limit order events.

trades Inferred trades (executions).

depth Order book price level depth through time.

depth.summary Limit order book summary statistics.

Author(s)

phil

Examples

```
## Not run:

csv.file <- system.file("extdata", "orders.csv.xz", package="obAnalytics")
lob.data <- processData(csv.file)

## End(Not run)
```

saveData *Save processed data.*

Description

Saves processed data to file.

Usage

```
saveData(lob.data, bin.file, ...)
```

Arguments

lob.data	lob.data data structure.
bin.file	File to save to.
...	saveRDS.

Details

Convenience function.

Author(s)

phil

Examples

```
## Not run:

saveData(lob.data, bin.file="/tmp/lob.data.rds", compress="xz")

## End(Not run)
```

`tradeImpacts` *Trade impacts.*

Description

Generates a data.frame containing order book impacts.

Usage

```
tradeImpacts(trades)
```

Arguments

`trades`	`trades` data.

Details

An impact consists of 1 or more limit orders being hit in order to fulfil a market order.

Value

A data.frame containing a summary of market order impacts:

id market order id

min.price minimum executed price

max.price maximum executed price

vwap VWAP obtained by market order

hits number of limit orders hit by market order

vol total volume removed by this impact

start.time (local) start time of this impact

end.time (local) end time of this impact

dir direction of this impact (buy or sell)

Author(s)

phil

Examples

```
# get impacts data.frame from trades data.
impacts <- tradeImpacts(lob.data$trades)

# impacts (in bps)
sell.bps <- with(impacts[impacts$dir == "sell", ], {
  (max.price-min.price)/max.price
})
10000*summary(sell.bps[sell.bps > 0])
```

trades *Trades.*

Description

Inferred trades (executions).

Format

A data.frame consisting of the following fields:

timestamp Local event timestamp.

price Price at which the trade occured.

volume Amount of traded volume.

direction The trade direction: *buy* or *sell*.

maker.event.id Corresponding market *making* event id in `events`.

taker.event.id Corresponding market *taking* event id in `events`.

maker Id of the market *making* limit order in `events`.

taker Id of the market *taking* limit order in `events`.

Details

The trades data.frame contains a log of all executions ordered by local timestamp. In addition to the usual timestamp, price and volume information, each row also contains the trade direction (buyer or seller initiated) and maker/taker limit order ids. The maker/taker event and limit order ids can be used to group trades into market impacts. See: `tradeImpacts`.

Author(s)

phil

See Also

Other Limit order book data: `depth.summary`, `depth`, `events`

Index