quanttrader

Release 0.5.0

Jul 04, 2023

Contents

Pure Python Backtest and Live Trade Package for Quant Traders

CHAPTER 1

Introduction

Welcome to quanttrader, a pure python-based event-driven backtest and live trading package for quant traders.

In most cases, a backtest strategy can be directly used for live trade by simply switching to live brokerage. A control window is provided to monitor live trading sessions for each strategy separately and the portfolio as a whole.

The source code is completely open-sourced here on GitHub. The package is published here on pypi and is ready to be pip installed. The document is hosted here on readthedocs.

This is NOT an ultra-low latency framework that can provide nano-second level executions. The response time, for example, between receiving data from the broker and sending out orders for a pairs-trading strategy that is subscribed to two stock feeds, is in the neighbourhood of milli-seconds. This package is designed mainly for quant traders who do not rely on market-making strategies.

Disclaimer: This is an open-source library that is free to use, free to contribute but use at OWN risk. It does NOT promise any future profits nor is responsible for any future loses.

CHAPTER 2

Installation

The quanttrader package is ready for pip install.

```
pip install quanttrader
```

To use source code, git pull the repository or use GitHub desktop and then add the project path to PYTHONPATH environment variable.

CHAPTER 3

Backtest

This document explains quanttrader backtest framework and code structure.

This repository contains examples of some classical strategies and their Sharpe ratios, as well as grid-search based parameter optimization. The backtest is designed to be working together with the pyfolio library.

One distinctive design in backtest is that it fills market order right away instead of filling against tomorrow's open price. After all, in the daily bar setting, it is better to send out order at 15:59:59 than waiting overnight for next day' open. If you disagree, simply save the market order similar to limit or stop order in the BacktestBrokerage class and then fill it on next tick.

Currently backtest accepts three data feeds.

- Daily bar or intraday bar from Yahoo Finance. See here for how to download.
- Historical intraday bar from Interactive Brokers. Use this script to download.
- Live tick recorded from live trading session. This video demonstrates how to do it in live session.

It is possible to load your own data source by following the above examples.

CHAPTER 4

Live Trading

This youtube video and accompanying document demonstrates step-by-step how to set up quanttrader for live trading. Currently quanttrader only supports Interactive Brokers.

Files used for live Trading are

- live_engine.py - the main entry point
- config_live.yaml - config file for live session
- instrument_meta.yaml - meta data for instruments to be traded
- prepare_trading_session.yaml - an example to demonstrate how to prepare data and strategy parameters before today's live session

CHAPTER 5

Instruments Supported

| Instrument | Example | | ——————- |:——————-:| | Stock | AMZN STK SMART| | Forex | EURGBP CASH IDEALPRO| | Futures | ESM9 FUT GLOBEX| | Options on Stock | AAPL OPT 20201016 128.75 C SMART| | Options on Futures | ES FOP 20200911 3450 C 50 GLOBEX| | Comdty | XAUUSD CMDTY SMART| | Stock Combo | SPY,AAPL BAG 756733 1 SMART 265598 1 SMART SMART| | Option Comb | SPY BAG 398943121 1 SMART 398943218 1 SMART SMART| | Calendar Spread | CL BAG 174230608 1 NYMEX 174230606 1 NYMEX NYMEX| | Inter-Comdty Spread | CL.BZ BAG 174230606 1 NYMEX 162929662 1 NYMEX NYMEX|

CHAPTER 6

Orders Supported

Basic order types. See IB Doc for details.

- Auction
- Auction Limit
- Market
- Market If Touched
- Market On Close
- Market On Open
- Market to Limit
- Limit Order
- Limit if Touched
- Limit on Close
- Limit on Open
- Stop
- Stop Limit
- Trailing Stop
- Trailing Stop Limit

CHAPTER 7

Brokerage

class quanttrader.brokerage.backtest_brokerage.**BacktestBrokerage**(*events_engine*, *data_board*)

Market order is immediately filled. Limit or stop order is saved to _active_orders for next tick

__init__(*events_engine*, *data_board*)

Initialize Backtest Brokerage.

Parameters

- **events_engine** – send fill_event to event engine
- **data_board** – retrieve latest price from data_board

cancel_order(*order_id*)

Handle cancel order request from client.

Parameters **order_id** – order id of the order to be canceled

Returns no return; cancel feedback is pushed into message queue

next_order_id()

Return next available order id for client to use.

Returns next available new order id

on_tick(*tick_event*)

Cross standing orders against new tick_event

Market order can be potentially saved and then filled here against tomorrow's open price

Parameters **tick_event** – new tick just came in

Returns no return; if orders are filled, they are pushed into message queue

place_order(*order_event*)

Place and fill client order; return fill event.

Market order is immediately filled, no latency or slippage the alternative is to save the orders and fill in on_tick function

Parameters **order_event** – client order received

Returns no return; fill_event is pushed into message queue

reset()
Reset Backtest Brokerage.

class quanttrader.brokerage.ib_brokerage.**InteractiveBrokers**(*msg_event_engine*, *tick_event_engine*, *account: str*)

__init__(*msg_event_engine*, *tick_event_engine*, *account: str*)
Initialize InteractiveBrokers brokerage.

Currently, the client is strongly coupled to broker without an incoming queue, e.g. client calls broker.place_order to place order directly.

Parameters

- **msg_event_engine** – used to broadcast messages the broker generates back to client
- **tick_event_engine** – used to broadcast market data back to client
- **account** – the IB account

cancel_all_orders()
Cancel all standing orders, for example, before one wants to shut down completely for some reasons.

cancel_historical_data(*reqid*)
Cancel historical data request. Usually not necessary.

Parameters **reqid** – the historical data request id

cancel_order(*order_id*)
Cancel client order.

Parameters **order_id** – order id of the order to be canceled

Returns no return. If order is successfully canceled, IB will return an orderstatus message.

connect(*host='127.0.0.1'*, *port=7497*, *clientId=0*)
Connect to IB. Request open orders under clientid upon successful connection.

Parameters

- **host** – host address
- **port** – socket port
- **clientId** – client id

static contract_to_symbol(*ib_contract*)
Convert IB contract to full symbol

Parameters **ib_contract** – IB contract

Returns full symbol

disconnect()
Disconnect from IB

heartbeat()
Request server time as heartbeat

static ib_order_to_order(*ib_order*)
Convert IB order to order event

Parameters **ib_order** – IB representation of order

Returns internal representation of order

log(*msg*)
Broadcast server log message through message queue

Parameters **msg** – message to be broadcast

Returns no return; log meesage is placed into message queue

next_order_id()
Return next available order id

Returns next order id available for next orders

static order_to_ib_order(*order_event*)
Convert order event to IB order

Parameters **order_event** – internal representation of order

Returns IB representation of order

place_order(*order_event*)
Place order to IB

Parameters **order_event** – client order to be placed

Returns no return. An order event is pushed to message queue with order status Acknowledged

reqCurrentTime()
Request server time on broker side

request_historical_data(*symbol*, *end=None*)
Request 1800 S (30 mins) historical bar data from Interactive Brokers.

Parameters

- **symbol** – the contract whose historical data is requested
- **end** – the end time of the historical data

Returns no returns; data is broadcasted through message queue

request_historical_ticks(*symbol*, *start_time*, *reqtype='TICKS'*)
Request historical time and sales data from Interactive Brokers. See here https://interactivebrokers.github.io/tws-api/historical_time_and_sales.html

Parameters

- **symbol** – the contract whose historical data is requested
- **start_time** – i.e. "20170701 12:01:00". Uses TWS timezone specified at login
- **reqtype** – TRADES, BID_ASK, or MIDPOINT

Returns no returns; data is broadcasted through message queue

setServerLogLevel(*level=1*)
Set server side log level or the log messages received from server.

Parameters **level** – log level

subscribe_account_summary()
Request account summary from broker

subscribe_market_data(*sym*)
Subscribe market L1 data. Market data for this symbol will then be streamed to client.

Parameters **sym** – the symbol to be subscribed.

subscribe_market_datas()
Subscribe market L1 data for all symbols used in strategies. Market data for this symbol will then be streamed to client.

subscribe_market_depth(*sym*)
Subscribe market L2 data. Market data for this symbol will then be streamed to client.

Parameters **sym** – the symbol to be subscribed.

subscribe_positions()
Request existing positions from broker

static symbol_to_contract(*symbol*)
Convert fulll symbol string to IB contract

TODO CL.HO BAG 174230608 1 NYMEX 257430162 1 NYMEX NYMEX # Inter-comdty ES.NQ BAG 371749798 1 GLOBEX 371749745 1 GLOBEX GLOBEX # Inter-comdty CL.HO BAG 257430162 1 NYMEX 174230608 1 NYMEX NYMEX

Parameters **symbol** – full symbol, e.g. AMZN STK SMART

Returns IB contract

unsubscribe_account_summary()
Stop receiving account summary from broker

unsubscribe_market_data(*sym*)
Unsubscribe market L1 data. Market data for this symbol will stop streaming to client.

Parameters **sym** – the symbol to be subscribed.

unsubscribe_market_depth(*sym*)
Unsubscribe market L2 data. Market data for this symbol will stop streaming to client.

Parameters **sym** – the symbol to be subscribed.

unsubscribe_positions()
Stop receiving existing position message from broker.

CHAPTER 8

Data

class quanttrader.data.backtest_data_feed.**BacktestDataFeed**(*start_date=None*, *end_date=None*)

BacktestDataFeed uses PLACEHOLDER to stream_next; actual data comes from data_board.get_hist_price This is an easy way to handle multiple sources

stream_next()
Place the next TickEvent into the event queue.

subscribe_market_data(*symbols=None*)
subscribe to market data

unsubscribe_market_data(*symbols=None*)
unsubscribe market data

class quanttrader.data.data_board.**DataBoard**

Data tracker that holds current market data info

get_current_price(*symbol*, *timestamp*)
Returns the most recent price for a given ticker based on current timestamp updated outside of data_board

get_hist_sym_time_index(*symbol*)
retrieve historical calendar for a symbol this is not look forwward

get_hist_time_index()
retrieve historical calendar this is not look forwward

get_last_price(*symbol*)
Returns last price for a given ticker because self._current_time has not been updated by current tick

get_last_timestamp(*symbol*)
Returns the most recent timestamp for a given ticker

class quanttrader.data.live_data_feed.**LiveDataFeed**(*events_queue*, *init_tickers=None*, *start_date=None*, *end_date=None*, *calc_adj_returns=False*)

Live DataFeed class

stream_next()
Place the next BarEvent onto the event queue.

subscribe_ticker(*ticker*)
Subscribes the price handler to a new ticker symbol.

CHAPTER 9

Event

class quanttrader.event.backtest_event_engine.**BacktestEventEngine**(*datafeed*)

Event queue + a while loop to dispatch events

put(*event*)

put event in the queue; call from outside

register_handler(*type_*, *handler*)

register handler/subscriber

run(*nSteps=-1*)

run backtest, if nSteps = -1, run to the end; else run nSteps

unregister_handler(*type_*, *handler*)

unregister handler/subscriber

class quanttrader.event.live_event_engine.**LiveEventEngine**

Event queue + a thread to dispatch events

put(*event*)

put event in the queue; call from outside

register_handler(*type_*, *handler*)

register handler/subscriber

start(*timer=True*)

start the dispatcher thread

stop()

stop the dispatcher thread

unregister_handler(*type_*, *handler*)

unregister handler/subscriber

CHAPTER 10

GUI

class quanttrader.gui.ui_account_window.**AccountWindow**(*account_manager*, *parent=None*)

update_table(*account_event*)
Only add row

class quanttrader.gui.ui_fill_window.**FillWindow**(*parent=None*)
present fills

update_table(*fill_event*)
Only add row

class quanttrader.gui.ui_log_window.**LogWindow**(*parent=None*)

update_table(*geneal_event*)
Only add row

class quanttrader.gui.ui_order_window.**OrderWindow**(*order_manager*, *broker*, *parent=None*)
Order Monitor

update_order_status(*order_id*)
This is called by fill handler to update order status

update_table(*order_event*)
If order id exist, update status else append one row

class quanttrader.gui.ui_position_menu.**PositionMenu**(*staretgy_manager*)

class quanttrader.gui.ui_position_menu.**PositionMenuBottom**(*strategy_manager*, *parent=None*)

class quanttrader.gui.ui_position_window.**PositionWindow**(*parent=None*)

class quanttrader.gui.ui_risk_menu.**RiskMenu**(*strategy_manager*)

class quanttrader.gui.ui_strategy_window.**StrategyWindow**(*strategy_manager*, *parent=None*)

Strategy Monitor

class quanttrader.gui.ui_trade_menu.**TradeMenu**(*broker*, *event_engine*, *order_manager*, *instrument_meta*)

place_order()

This is not tracked by strategy_manager; tracked by global order_manager :return:

CHAPTER 11

Order

class quanttrader.order.order_manager.**OrderManager**(*name='Global'*)

Manage/track all the orders

on_cancel(*oid*)

This proactively set order status to PENDING_CANCEL :param o: :return:

on_fill(*fill_event*)

on receive fill_event from broker

on_order_status(*order_event*)

on order status change from broker including canceled status

on_tick(*tick_event*)

CHAPTER 12

Performance

class quanttrader.performance.performance_manager.**PerformanceManager**(*instrument_meta*)
https://www.quantopian.com/docs/api-reference/pyfolio-api-reference Record equity, positions, and trades in accordance to pyfolio format First date will be the first data start date

update_performance(*current_time*, *position_manager*, *data_board*)
update previous time/date

CHAPTER 13

Position

CHAPTER 14

Risk

class quanttrader.risk.risk_manager.**PassThroughRiskManager**

order_in_compliance(*o*, *strategy_manager=None*)

Pass through the order without constraints :param original_order: :param env: e.g. strategy_manager that stores order info vs config info :return:

class quanttrader.risk.risk_manager.**RiskManager**

order_in_compliance(*o*, *strategy_manager=None*)

Parameters

- **original_order** –
- **env** – straegy_manager

Returns

CHAPTER 15

Strategy

class quanttrader.strategy.strategy_base.**StrategyBase**

Base strategy class

adjust_position(*sym*, *size_from*, *size_to*, *timestamp=None*)

use market order to adjust position :param sym: :param size_from: :param size_to: :param timestamp: used by backtest broker to get price on timestamp :return:

cancel_all()

cancel all standing orders from this strategy id :return:

cancel_order(*oid*)

on_fill(*fill_event*)

on order filled derived class call super().on_fill first

on_init(*strategy_manager*, *data_board*, *instrument_meta*)

on_order_status(*order_event*)

on order acknowledged :return:

on_start()

on_stop()

on_tick(*tick_event*)

Respond to tick

place_order(*o*)

expect user to set up order type, order size and order price

set_capital(*capital*)

set_name(*name*)

set_params(*params_dict=None*)

set_symbols(*symbols*)

class quanttrader.strategy.strategy_manager.**StrategyManager**(*config*, *broker*, *order_manager*, *position_manager*, *risk_manager*, *data_board*, *instrument_meta*)

cancel_all()

cancel_order(*oid*)

cancel_strategy(*sid*)
call strategy cancel to take care of strategy order_manager

flat_all()
flat all according to position_manager TODO: should turn off all strategies? :return:

flat_strategy(*sid*)
flat with MARKET order (default) Assume each strategy track its own positions TODO: should turn off strategy?

load_strategy(*strat_dict*)

on_cancel(*order_event*)
TODO no need for this

on_fill(*fill_event*)
assign fill ordering to order id ==> strategy id TODO: check fill_event source; if not, fix it or use fill_event.order_id

on_order_status(*order_event*)
TODO: check if source is working :param order_event: :return:

on_position(*pos*)
get initial position read from config file instead :param pos: :return:

on_tick(*k*)

pause_strategy(*sid*)

place_order(*o*, *check_risk=True*)

start_all()

start_strategy(*sid*)

stop_all()

stop_strategy(*sid*)

CHAPTER 16

Trading Engine

class quanttrader.gui.ui_main_window.**MainWindow**(*config*, *instrument_meta*, *strat_dict*)

closeEvent(*self*, *a0: QCloseEvent*)

connect_to_broker()
Connect to broker :return: None

disconnect_from_broker()
Disconnect from broker :return: None

init_central_area()

init_menu()

init_status_bar()

liquidate_all_strategy()

liquidate_strategy()

open_position_widget()
Open position monitor for strategies :return: None

open_risk_widget()
Open risk manager monitor for strategies :return: None

open_trade_widget()
Open discretionary trade window :return: None

save_orders_and_trades()

start_all_strategy()

start_strategy()

stop_all_strategy()

stop_strategy()

update_status_bar(*message: str*)
Update status bar with message :param message: message to be shown in the status bar :return: None

class quanttrader.gui.ui_main_window.**StatusThread**

run(*self*)

status_update

class quanttrader.backtest_engine.**BacktestEngine**(*start_date=None*, *end_date=None*)
Event driven backtest engine

add_data(*data_key*, *data_source*, *watch=True*)
Add data for backtest :param data_key: AAPL or CL :param data_source: dataframe, datetimeindex :param watch: track position or not :return:

run()
Run backtest

set_capital(*capital*)
set capital to the global position manager

set_instrument_meta(*instrument_meta*)

set_strategy(*strategy*)

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Python Module Index

q

Index

Symbols

A

B

C

D

F

P

Q

R

T

U