etc

Introduction to the challenge

etc is a programming challenge. The goal is to write a program that will make fake money by trading securities on a fake financial exchange. The market participants will include other people's programs, as well as some programs provided by us (the "marketplace bots"). To start, read over this brief intro to finance to learn the basic financial concepts and trading strategies you will need for this competition.

Table of Contents

Getting started

Developing your bot

We will be running this challenge on Amazon EC2.

Each team will receive one Ubuntu Linux box in the cloud which is yours and yours alone, which we'll call your "bot box".

When we are ready to begin, we will tell you the (public) IP address of your bot box. To get started, log in to the box by opening a terminal on your laptop and using ssh:

```
ssh ubuntu@54.1.2.3     # Replace with your bot box's public IP
```

NOTE: You will need to run your trading bot from the bot box directly. If you're developing on your local machine, you can copy your code to your bot box is with the command "scp". Here is an example:

```
# Replace 54.1.2.3 with your bot box's public IP
scp ~/amp/sample-bot.py ubuntu@54.1.2.3:~/sample-bot.py
```

Make sure to copy your code over whenever you make changes.

Do not remove our SSH authorized keys on your bot box

We need to be able to SSH into the bot boxes in order to connect them to the production exchanges. If you remove us, your bot will not be able to compete. If all you do is SSH to your box, change bot code, and run your bot, then you will not need to worry about this.

Testing your bot

You have access to three test exchanges where you can test out your bot. The three exchanges run slightly different versions of the marketplace. On the “prod-like” test exchange, the marketplace runs at the true competition speed and simulates real rounds by restarting every five minutes. On the “slower” test exchange, the marketplace runs more slowly. On the “empty” test exchange, there is no marketplace – the exchange is empty except for your team.

The same symbols are available on the test exchanges as on the prod exchanges. Each round will have the same starting prices for all of the symbols, but the price path after that is random – you should not expect them to remain consistent between the test and prod exchanges, nor even between two different games on a prod exchange.

The slow test exchange is generally better for testing. The production exchange and prod-like exchanges are much faster, which can become hard to debug if you have a lot of print statements.

Connecting to the exchange

Below are links to sample code that will help you connect to the exchange. You can use the provided functions to connect the exchange and to send/receive messages. The technical details surrounding the exchange protocol are here.

Each team may have at most one connection open to an exchange at any time, but one team can connect to their three testing exchanges and the prod exchange at the same time.

Here is sample code the we recommend you use to start your bot:

- Python

Monitoring your bot

You can see the current books on test exchanges, as well as all positions and the history of trades. Use your team name and password to log in.

http://americas.etc.janestreet.com:8080

At this point, you should connect connect to your bot box, run your code against the slow test exchange, and observe the exchange state at the above URL.

```
# run this on your bot box
~/sample-bot.py --test slower
```

The default observer view is the “Positions & Trades” tab. You can also view the book for each symbol.

The starter code will send an order to buy BOND for 990 at startup. Navigate to / click on the “BOND” tab at the top of the page and confirm that you see your order (your “bid”). Note that it is on the bottom left side of the book. Why is this? If you don’t understand why, check with your TA!

NOTE: you will only see observer information for test exchanges once you connect. You will not see any information for production exchanges. This is intentional, and in real life you generally won’t have access to the internal books & positions on exchanges like NYSE.

The challenge structure

The challenge consists of a series of five minute matchups. In each matchup, you and another team compete to be the best market maker, providing liquidity to a third bot called the “marketplace”.

The marketplace has an internal random process for each stock which represents the fair value of that stock. (At the end of the matchup, the portfolio you have amassed will be evaluated using these fair values – see below). The marketplace will send some orders around the fair value. Some orders will, like yours, be orders intended to provide liquidity. Other orders from the marketplace will be emulating liquidity demanders – people who just wish to buy or sell a certain amount of shares, and don’t care too much about the price they get.

In particular, it is not the marketplace’s job to make money. Although technically speaking the marketplace is a “bot”, its positions or pnl is not very relevant. The marketplace is meant to represent a whole bunch of different market participants, some of whom are more price-sensitive than others. This is representative of the fact that we mentioned in the finance intro – it’s unhelpful to think of trading as a zero-sum game.

You can choose to participate, or not, in a given matchup, or join after the matchup begins, or give up halfway through – whatever you want.

At the end of a matchup, the exchange closes and then reopens shortly afterwards for the next matchup, where you’ll be playing alongside a different team. We reset your cash and positions too. You’ll want to make sure your bot can reconnect automatically when the exchange re-opens for the next matchup.

Your portfolio

You have an account (like a bank account) that can hold both currency and shares in financial securities. At the beginning of each matchup your account is empty – no currency, no shares. The number of shares of a particular security that you own is called your /position/ in

that security. If you buy a share by paying a currency, your position in that currency goes down and your position in that stock goes up. If you sell, the opposite. In this competition, the only currency we'll be dealing with is USD (US Dollars). Since USD is the only currency we'll be using, we may refer to it as "cash" in this handout.

Note that it is possible to have a negative position in any product including cash – you can effectively borrow money or shares from the bank at which you hold the account.

Risk limits

Your position in each security will be limited, with the limits depending on which stock or ETF you're trading – see the table below. For example, in BOND, the position limit is 100 shares; that means your position in BOND may never be less than -100 or greater than +100. In the real world, our positions are limited by the availability of stock borrow, capital constraints, and crash risk. Obviously, the greater your position, the more exposed you are to a crash in this security.

symbol	limit
BOND	100
VALBZ	10
VALE	10
GS	100
MS	100
WFC	100
XLF	100

These are enforced by the exchange: if you send an order such that if it and all your other orders in the same symbol and direction were fully filled it would put you over your position limits, then your order will be rejected.

There are no direct limits on your cash balance in either direction.

However, there are limits on your profit/loss. If at some point you get to a stage where your current pnl is too negative, you will be determined to have "blown out". We'll stop your participation in that matchup there, and close any positions you have at the fair value at that time. The limit is -30,000. You will not be able to reconnect until the next round.

If your bot blows out, it likely means that it did a lot of really bad trades really quickly. If this happens, it's a good idea to check with your TA if you don't understand what caused your bot to behave this way.

Profit and loss

Your profit/loss (aka “pnl”) can be computed by taking the amount of cash in your account (e.g. if you buy a security for 10, and then sell it for 12, you would have 2 cash in your account), and adding the value of all your holdings (again, as determined by the marketplace’s internal notion of fair value). The main goal is to maximize your pnl over all the rounds you participate in. Another important goal is to avoid doing very poorly in any given round. This is reflective of the real world: you may do particularly well on some days, but you really want to avoid losing all your money on any given day because then you’re out of a job.

Rounds

A round will last for five minutes, after which the exchange will shut down. A new exchange will shortly start up and be available on the same address.

We don’t carry your positions forward from one game to the next.

Be careful – you should not assume that the values of securities in one game will be anywhere near the values in a previous game.

If your bot quits once its connection to the exchange goes down, you might want to run your bot in a bash loop so that it automatically reconnects:

```
while true; do ./sample-bot.py; sleep 5; done
```

Running in production

When you’re ready to run your bot in production, you can run run your bot as follows:

```
# run on your bot box
~/sample-bot.py --production
```

You can see the results from your previous production round at this url. Replace “TEAMNAME” with your team name.

http://americas.etc.janestreet.com:8080/go/last-round-results/TEAMNAME

Some golden rules

1. You must not attempt to manipulate other people’s bots.
2. You must not attempt to manipulate the exchange.
3. Your bot should be entirely written by your team over the course of this ETC (standard libraries are fine).
4. Don’t do anything you wouldn’t do on a real exchange.

You are building a bot to provide a service to the marketplace bots, and attempting to do so in the best possible way.

Some examples of what would count as manipulation:

- Trading with yourself in order to publish “fake” trades to all participants, in an attempt to have them believe the trade price is a reasonable fair value for the security.
- Sending orders to the exchange where you know at the time of sending that you want to cancel the orders and never let them trade. If you think it would be a bad idea to buy security for price P, then you shouldn’t send orders to buy that security for price P.

Don’t download a bot from the internet and run that instead of writing a bot. Or other silly things like that. Basically, don’t be a jerk.

We reserve the right to subtract some from your bot’s profit, or disqualify you completely, if we feel you’ve violated these rules. Obviously they are somewhat vague. Please come talk to us if you have any concerns. If you find a bug in the exchange, please tell us. If you think someone else has been deliberately manipulating your bot, please tell us too.

Technical details

The exchange protocol

You and the exchange send a variety of different message types to each other.

Outgoing Messages

You can send the following kinds of messages to the exchange

- Hello: the first message you must send, identifying yourself to the exchange
- Add: a request to buy or sell a security; “add order”
- Cancel: a request to remove from the book some previously-placed order that hasn’t traded yet
- Convert: a request to convert an ETF or ADR (from or to its underlying basket)

Your starter code already sends a Hello message. You can send other outgoing messages using these using the following helper methods:

```
exchange.send_add_order(order_id:int, symbol:str, dir:Dir,
price:int, size:int)
exchange.send_cancel_msg(order_id:int)
exchange.send_convert_message(order_id:int, symbol:str, dir:Dir,
size:int)
```

Incoming Messages

The exchange can send a variety of messages back to your bot

- Hello: the first message the exchange will send you when you connect, containing your positions
- Ack: “your order was successfully placed on the book” (this does not mean it traded!)
- Reject: “your order wasn’t valid for this reason: …” (e.g. negative price, malformed syntax etc.)
- Error: an error related to your bot that’s not associated with a specific order
- Out: following a cancel or once your order is completely filled, “your order is no longer on the book”
- Fill: “your order traded”
- Book: “the current state of the book is…”
- Trade: “two (anonymous) people traded at price X”
- Open: “the market for a specific security has opened”
- Close: “the market for a specific security has closed”

Incoming messages from the exchange are a dictionary with a “type” key and some number of other keys based on the value associated with the “type” key.

Your starter code reads the message stream from the exchange in a while loop – do not change this!! If you want to inspect information about the exchange, you should look at the keys in the “message” variable at the top of the while loop. Exchange messages have the following possible keys and values:

```
{'type':'hello', 'symbols':[{'symbol':'SYM', 'position':N},
...]}
{'type':'open', 'symbols':['SYM1', 'SYM2', ...]}
{'type':'close', 'symbols':['SYM1', 'SYM2', ...]}
{'type':'error', 'error':'MSG'}
{'type':'book', 'symbol':'SYM', 'buy':[[PRICE,SIZE], ...],
'sell':[...]}
{'type':'trade', 'symbol':'SYM', 'price':N, 'size':N}
{'type':'ack', 'order_id':N}
{'type':'reject', 'order_id':N, 'error':'MSG'}
{'type':'fill', 'order_id':N, 'symbol':'SYM', 'dir':'BUY',
'price':N, 'size':N}
{'type':'out', 'order_id':N}
```

Message contents obey the following rules

- N: int
- SYM: uppercase string symbol name, like BOND or VALE
- DIR: A direction is one of Dir.BUY or Dir.SELL
- PRICE, SIZE: Prices and sizes are *positive* ints strictly less than 1000000.
- Positions are ints (at least smaller than 2^32)
- Order ids are non-negative ints (at least smaller than 2^32)

- MSG: Arbitrary strings potentially including spaces (but they won't contain newlines or nulls or anything tricksy).

Handshake

You must send a Hello message at first, and the first message from the exchange is also Hello. Your starter code already does this.

You will immediately receive a "hello" from the exchange (or an error). The hello will include your current positions in each symbol (including USD). At the start these will all be zero; these numbers are only useful to you if you crash and reconnect.

The starter code already sends and receives the first Hello-Hello exchange. The following code snippet prints the output at startup:

```
hello_message = exchange.read_message()
print("First message from exchange:", hello_message)
```

So, when you run your bot, you'll see something like this:

```
First message from exchange:
{'type': 'hello', 'symbols': [{'symbol': 'BOND', 'position': 0},
...]}
```

Orders

You, the client, must pick unique identifiers for your orders and send them along with the rest of the details in the Add message. Ack, reject, fill and out messages will refer to the order by this identifier. It need not be globally unique, merely unique to your connection to the exchange. Besides these restrictions, you may pick them however you like, remembering that they must be non-negative integers.

To, say, enter a buy order for BOND with price 1002 and size 50, issue:

```
exchange.send_add_order(order_id=5, symbol='BOND', dir=Dir.BUY,
price=1002, size=50)
```

You will receive the following message contents, either:

```
{'type':'ack', 'order_id':5}
```

or:

```
{'type': 'reject', 'order_id': 5, 'error': 'REASON'}
```

where 'REASON' might be one of the strings mentioned above.

Suppose that your order could trade immediately with a “sell 10 at $1001” order that was resting on the market. Shortly after the ack you would receive a Fill message with the following contents:

```
{'type': 'fill', 'order_id': 5, 'symbol': 'BOND', 'dir':
Dir.BUY, 'price': 1001, 'size': 10}
```

The fill contains a reminder of the order’s symbol and direction, the price that the order traded at, and the number of shares that traded.

Note that your order with order_id=5 is still on the market, but now its size is 40.

When your order has been fully filled, you will also receive an Out message. Continuing the above example, if a little later on, someone else submitted a large low sell order, you would eventually receive the following 2 messages:

```
{'type': 'fill', 'order_id': 5, 'symbol': 'BOND', 'dir':
Dir.BUY, 'price': 1002, 'size': 40}
{'type': 'out', 'order_id': 5}
```

Note that at roughly the same time as you receive a Fill message, you will also receive a Trade message.

To instead cancel this order, issue

```
exchange.send_cancel_msg(order_id=5)
```

The reply will be an Out or an Error:

```
{'type': 'out', 'order_id': 5}
```

Provided you have completed the handshake, “cancel” messages never fail: you will always receive an “out” reply, even if the server does not recognise that order. If a cancel and a fill race (and the cancel loses), you might receive two outs for the same order id. Furthermore, you might receive an out for an order that was neither cancelled by you nor fully filled (an “unsolicited out”), if the exchange decides it wants to cancel your order for some specific reason. You may wish to ignore outs you don’t recognise.

Conversions to and from an ADR or ETF are done using the Convert message, which is similar to Add but without a price:

```
# note: ADR conversions only work if you use symbol='VALE' and
the
# following code will not work if you try to run it with VALBZ
exchange.send_convert_message(order_id:4, symbol:'VALE',
dir:Dir.BUY, size:10)
```

Dir.BUY means you receive the ADR/ETF and give out the constituent basket of stocks. Dir.SELL means you give out the ADR/ETF and receive the constituent basket of stocks. In both cases, you pay a conversion fee. You are not allowed to convert if it would put you over any position limit – in the ADR/ETF or in the stocks. The order ID for conversions shares a namespace with orders.

The reply will be either an Ack message or a Reject message indicating whether the conversion was successful or not. It is not possible for a conversion to partially succeed.

Public feed

Trade and book messages constitute the anonymised public feed. Contents of a Trade message look like:

```
{'type':'trade', 'symbol':'BOND', 'price':1002, 'size':40}
```

means that 40 shares of BOND have traded at price $1002 between two people (or one person with themselves).

```
{'type':'book', 'symbol':'BOND', 'buy':[[999,12],[998,100],
[995,1]], 'sell':[[1001,4],[1002,15],[1003,100]]}
```

means that one or more people are willing to buy BOND for $999, and the sum of the sizes of those orders is 12. (Willing to buy 100 shares of BOND for $998, etc.) On the other side, there are people willing to sell up to 4 shares at $1001.

The book will be truncated to 15 price levels. It will be sorted most-aggressive-first (i.e. bids are high-to-low, offers are low-to-high), so that the best bid and offer are in the location indicated by “BEST” below:

```
{'type':'book', 'symbol':'BOND', 'buy':[BEST, ...], 'sell':
[BEST, ...]}
```

There are no guarantees about when you’ll receive these messages in relation to the private ones. Notably:

- when you add an order, the “book” update to include your order may arrive before or after the ack (moreover, not necessarily immediately before or after)
- similarly for trade & fill messages.

Errors

“Error” and “reject” messages are the server complaining about your bot. If the server is able to associate the error with an “add” message, then it will send a “reject” with the order id that failed and the error message. Otherwise, you will simply receive an “error”.

If the server detects that you have disconnected then all of your open orders will be canceled.

Available symbols

The following symbols are available:

- BOND
- VALBZ
- VALE
- GS
- MS
- WFC
- XLF

The symbol BOND has a marketplace fair value of precisely 1000 at all times, but could possibly trade at other prices.

VALBZ is a regular stock.

The symbol VALE is an ADR of VALBZ (you can think of this as an ETF with one component). 1 share of VALE can be converted to/from 1 share of VALBZ. The conversion fee for VALE is a fixed cost of 10 per conversion (regardless of the number of shares converted).

Symbols GS, MS, and WFC are regular stocks.

The symbol XLF is an ETF. 10 shares of XLF can be converted to/from a basket of:

- 3 BOND
- 2 GS
- 3 MS
- 2 WFC

You must convert XLF in multiples of 10. The conversion fee for XLF is a fixed cost of 100 per conversion (regardless of the number of shares converted). Note that to convert 10 XLF shares, the size in your “convert” message should be 10, not 1.

Advice from the devs

Keep it simple

Most of the time, a complicated algorithm is hard to think about and hard to write.

Start trading early

It’s a good idea to find a strategy to make you money, and continue using it throughout the competition. You should try to provide liquidity early.

If something seems like it should be solved for you already, it probably is

In past competitions, people have spent lots of time doing low-level things like writing line parsers for the TCP protocol. Every language that's been used in past etc events has had the native ability, or had libraries readily available to read whole lines from a socket.

Spend some time to make sure you can debug your bot

When there's a problem with your bot, make sure you can figure out what's going wrong. Many languages have built-in logging frameworks or great 3rd party libraries for these. Printf can also be your friend.

Advice from the traders

Step 1: trading BOND

BOND should be pretty straightforward to trade for positive expectancy, since its fair value is known! Buy it for less than 1000 or sell it for more than that and you're golden. Fortunately, there's a lot more of interest to come.

Step 2: trading the VALBZ/VALE ADR pair

These are two stocks for which the fair value changes over time but is always equal across the two. VALBZ is the more liquid side, and so you'll notice that it's a better source of price information, but it's a lot harder to make money trading. Protip: use the more liquid leg (VALBZ) to price out and trade the less liquid leg (VALE). The position bounds are tight, though, so you'll have to convert and trade both stocks if you want to maximize your expected profits.

Step 3: the XLF ETF

Now instead of a one-to-one relationship, you'll need to price out a basket of stocks in order to trade XLF. Again, due to the position bounds, careful position management (and converting) and good execution methods will be critical to success.

Get going!

The earlier you can get something simple that works out into the marketplace, the more time your bot can spend making profits. Work on version 2.0 while you're making money with version 1.0.

Maximize!

Keep tuning and improving your algorithms; maybe a simple change to one of your parameters could double your profits per round. Use the

test exchange to experiment; the marketplace bots there are exactly the same as on the competition exchange.

FAQ

My messages are being ignored!

You need to send a newline after each message (even when using the JSON protocol).

Can I use <favorite library> or <favorite search engine>?

You are welcome (and encouraged!) to search for solutions to problems, standard libraries etc. online. But please don’t download bot code, post solutions online, get someone to write code for you, or anything like that.

Quiz

To get started, complete a quick quiz and then come see us to get connection details for your box.