HFT Simulation System
Part III: Observations

Elaine Wah
ewah@umich.edu

Updated: June 2013

1 Introduction

Observations are created each time a simulation is run. The format of the observation files are dictated by the requirements for our EGTAOnline testbed. They are created as JSON files, and there are two types of observation files (one for each use case).

2 Reading an observation file

Observations are saved in the format `observation#.json`, where # is the observation number (from 1 to the max number of samples gathered). Note also that this number is the second input argument to the simulator. As such, be sure to save different simulation runs in different directories; otherwise you run the risk of overwriting previous observations.

Depending on the use case (**EGTA** vs. **Market models**), the observation file will differ. Parsing observation files (and converting to CSV) is currently only supported for the Market models use case.

2.1 EGTA

In the **EGTA** use case, there is a section in the json file called "players" containing the player's role, strategy, payoff (i.e., surplus), and any player-specific features:

```
{"players":[
  {
    "payoff":0,
    "strategy":"ZIR:bidRange_1000",
    "role":"BACKGROUND"
  },
  {
    "payoff":121208,
    "strategy":"BASICMM:sleepTime_500",
    "role":"MARKETMAKER"
  }],
 ...
```

The other section in the json file will be "features" which will include information on the simulation configuration and aggregate statistics computed at the end of the simulation. This is discussed in depth in the following section.

2.2 Market models

In the **Market models** use case, there will be no player-level data (as there are no players). The features section will be the same as in the EGTA use case. Features are added separately for each model and thus labeled as such.


```
...
"features":
 {"":
   {"shock_var":1.5E8,
    "sim_length":15000,
    "nbbo_latency":200,
    "ZIP_num":0,
    ...
   },
  "twomarketla_vol":
   {"freq250_mean_stdprice":17191.8795,
    "freq250_std_price_mkt1":9182.092,
    "freq250_mean_logprice":9.6298,
    "freq250_std_price_mkt2":25201.667,
    "freq250_std_logreturn_mkt2":0.7307,
    "freq250_std_logreturn_mkt1":0.0761,
    "freq250_mean_logreturn":0.4034
   },
  "twomarketla_spreads":
   {"med_nbbo_up_to_maxtime":48881.0,
    "med_mkt2_up_to_maxtime":33122.0,
    "med_mkt1_up_to_maxtime":48881.0,
    "mean_mkt_up_to_maxtime":41001.5
   },
  "twomarketla_surplus":
   {"sum_background_nodisc":-390399.0,
    "sum_total_disc6.0E-4":-175570.6139,
    "sum_hft":60579.0,
    "sum_background_disc6.0E-4":-236149.6139,
    "sum_marketmaker_disc6.0E-4":"NaN",
    "sum_total_nodisc":-329820.0,
    "sum_marketmaker_nodisc":"NaN"
   },
  "twomarketla_trans":
   {"price_mean":91012.525,
    "rmsd":37443.4722,
```

```
     "zi_num":66,
     "price_std":22348.1091,
     "freq250_rmsd":52523.3124,
     "la_num":14
    },
   "twomarketla_exectime":
    {"mean":85.275}
  }
}
```


3 Adding observations & features

Each agent has a `getObservation` method for generating the player-level observation. This method will only be called if the agent is created as a player.

All features are created as `Feature` objects, which contains methods for adding various metrics (e.g., mean, variance, max, min, median, etc.). The methods in the `Feature` class use `DescriptiveStatistics` in order to compute these statistics, and in these computations $NaNs$ will be included, so be careful about instances in which market performance measures are undefined.

4 Parsing observation files

When observation files are parsed, the JSON headers are flattened (in the JSON file itself, each observed feature is stored as a hierarchical object, so flattening is necessary to view the data in a CSV). To parse a directory of observations, use the following command:

```
./parse_single.sh <CSV> <directory of observations>
```

For example, `./parse_single.sh test.csv simulations/test` will parse all observations in the `simulations/test` directory.

Note that the parser will only generate column headers if the output CSV does not yet exist; if you are not seeing header files, delete the CSV and re-parse.

5 Guide to feature names

Descriptions for the various types of features (statistics) gathered are in Table 1. Model names have been removed for the sake of brevity, but a typical feature, after flattening the JSON observation files, will have a name with the following format:

```
<model name><model config>_<feature name>
```

For example, the label for average transaction prices in the `TWOMARKET` model with a single latency arbitrageur would be:

```
twomarketla_trans_price_mean
```

If you see NaNs, that typically means that there were no values for the specific metric, so the value is undefined. Max time is computed dynamically within the `Observations` class (it's based on when all agents will have arrived, on average), and sampling frequencies are hard-coded within `Consts`.

NAME (FLATTENED)	DESCRIPTION
spreads_med_mkt#_up_to_maxtime	median spread in market -#, using values up to some maximum time (computed dynamically and listed in the configuration section of the observation file)
spreads_med_mean_up_to_maxtime	average median spread over all markets
spreads_med_nbbo_up_to_maxtime	median spread of NBBO
vol_freqX_mean_stdprice	average volatility over all markets, measured by standard deviation of midquote prices sampled every X timesteps
vol_freqX_mean_logprice	average volatility over all markets, measured by log of standard deviation of midquote prices sampled every X timesteps
vol_freqX_mean_logreturn	average volatility over all markets, measured by standard deviation of log returns sampled every X timesteps
vol_freqX_std_price_mkt#	volatility in market -#, measured by log of standard deviation of midquote prices sampled every X timesteps
vol_freqX_std_logreturn_mkt#	volatility in market -#, measured by standard deviation of log returns sampled every X timesteps
trans_price_mean	average transaction price
trans_rmsd	root mean square deviation (RMSD) between transaction prices and the value of the fundamental at the time of execution
trans_freqX_rmsd	RMSD based on prices sampled every X timesteps
trans_price_std	standard deviation of transaction prices
<agent type>_num	number of transactions by the given agent type (e.g., ZI, ZIR, LA)
exec_time_mean	average time between when bid is submitted to when it transacts
surplus_sum_<role>_nodisc	total raw (undiscounted) surplus for agents in the given role (e.g., background, HFT); if value is NaN, then there were no transactions recorded for agents in this role
surplus_sum_total_nodisc	total raw surplus for all agents
surplus_sum_<role>_discY	total surplus (discounted by Y) for agents in the given role
surplus_sum_total_discY	total surplus (discounted by Y) for all agents

Table 1: List of features for a given model.