REINFORCEMENT LEARNING TECHNIQUES AND OPTIMAL EXECUTION

D. NARAYANA

1. Introduction

Markov Decision Processes are a tool for modelling sequential decision making problems where a decision maker interacts with a system in a sequential fashion. Given an MDP $\mathcal{M}$, this interaction happens as follows:

Let $t \in \mathbb{N}$ denote the current time, let $X_t \in \mathcal{X}$ and $A_t \in \mathcal{A}$ denote the random state of the system and the action chosen by the decision maker at time t, resp. Once the action is selected, it is sent to the system which makes a transition:

$$(X_{t+1}, R_{t+1}) \sim P_0(.|X_t, A_t) \tag{1}$$

In particular, X_{t+1} is random and $P(X_{t+1} = y|X_t = x, A_t = a) = P(x, a, y)$ holds for any $x, y \in \mathcal{X}$, $a \in \mathcal{A}$. Further, $\mathbb{E}[R_{t+1}|X_t, A_t] = r(X_t, A_t)$. The decision maker then observes the next state X_{t+1} and reward R_{t+1}, chooses a new action $A_{t+1} \in \mathcal{A}$ and the process is repeated.

The goal of the decision maker is to come up with a way of choosing the actions so as to maximize the expected total discounted reward

The *return* underlying a behavior is defined as the total discounted sum of the rewards incurred:

$$R = \sum_{t=0}^{\infty} \gamma^t R_{t+1}$$

Thus if $\gamma < 1$, then rewards far in the future woth exponentially less than the reward received at the first stage. An MDP when the return is defined by this formula is called a *discounted reward* MDP. When $\gamma = 1$, the MDP is called *undiscounted*.

The goal of the decision maker is to choose a behavior that maximizes the expected return, irrespctively of how the process is started. Such a maximizing behaviour is said to be *optimal*.

In reinforcement learning, a learning agent wanders in an unknown environment and tries to maximize its long term return by performing actions and receiving rewards. One the challenges in reinforcement learning is to understand how current action will affect future rewards. A good way to model this task is with Markov Decision Processes (MDP), which have become the dominant approach in reinforcement learning.

An MDP consists of states, actions, and for each state-action pair, a distribution of next states (the states reached after performing the action in the given state). In addition, there is a reward function that assigns a stochastic reward for each state and action. The *return* combines a sequence of rewards into a single value that the

March 9, 2019

agent tries to maximize. A discounted *return* has a parameter $\gamma \in (0,1)$, where the reward received at step k is discounted by γ^k

One of the challenges of reinforcement learning is when the MDP is not known, and we can only observe the trajectory of states, actions and rewards generated by the agent wandering in the MDP. There are two basic conceptual approaches to the learning problem:

First approach: This is model based. We first need to reconstruct a model of the MDP, and then find an optimal policy for the approximate model

Second approach: This consists a set of implicite methods. Here we update the information after each step, and based on this derive an estimate to the optimal policy. One of the most popular implicite methods is Q-learning.

Q-learning is an off-line policy method that can be run on top of any strategy wandering in the MDP. It uses the information observed to approximate the optimal function, from which one can construct the optimal policy. There are various proofs that Q-learning does converge to the optimal Q function, under very mild conditions.

Give latest references for this convergence, may be exact theorems from Powell or Gosavi books

. The conditions have to do with exploration policy and learning rate. For the exploration, each state-action pair needs to be performed infinitely many times. The learning rate controls how fast we modify our estimates. One expects to start with a high learning rate, which allows fast changes, and lowers the learning rate as time progresses. The basic conditions are that the sum of the learning rates goes to infinity(so that any value could be reached) and that the sum of the squares of the learning rates is finite (which is required to show that the convergence is with the probability one)

Present above text with equations. May be refer to Ross book

Our model is a synchronous model, where all state-action pairs are updated simultaneously.

2. Model

We define a Markov Decision Process (MDP) as follows:

Definition: A Markov Decision Process (MDP) M is a 4-tuple (S,U,P,R), where S is a set of the states, U is a set of actions (U(i) is the set of actions available at state i), $P^M_{i,j}(a)$ is the transition probability from state i to state j when performing action $a \in U(i)$ in state i, and $R^M(s,a)$ is the reward received when performing action a in state s.

We assume that $R^M(s,a)$ is bounded by R_{max}, i.e., $\forall\, s,a,\ 0 \le R^M(s,a) \le R_{max}$.

List down conditions on $P^M_{i,j}(a)$ and $R^M(s,a)$?

A strategy for an MDP assigns, at each time t, for each state s, a probability for performing action $a \in U(s)$, given a history $F_{t-1} = \{s_1, a_1, r_1, \cdots, s_{t-1}, a_{t-1}, r_{t-1}\}$ which includes the states, actions and rewards observed until time t-1. A policy is memory-less strategy, i.e., it depends only on the current state and not on the history. A deterministic policy assigns each state a unique action.

While following a policy π, we perform at time t action a_t at state s_t and observe a reward r_t (distributed according to $R^M(s,a)$), and the next state s_{t+1} (distributed

according to $P^M_{s_t,s_{t+1}}(a_t)$). We combine the sequence of rewards to a single value called the return, and our goal is to maximize the return. In our alogirthm, we focus on discounted return, which has a parameter $\gamma \in (0,1)$, and the discounted return of policy π is

$$V^\pi_M = \sum_{t=0}^{\infty} \gamma^t r_t$$

where r_t is the reward observed at time t. Since all rewards are bounded by R_{max}, the discounted return is bounded by $V_{max} = \frac{R_{max}}{1-\gamma}$.

We define a value function for each state s, under policy π, as $V^\pi_M(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i]$, where the expectation is over a run of policy π starting at state s. We define a state-action value function

$$Q^\pi_M(s,a) = R_M(s,a) + \gamma \sum_{\overline{s}} P^M_{s,\overline{s}}(a) V^\pi_M(\overline{s}) \tag{2}$$

whose value is the return of initially performing action a at state s and then following policy π

Let π^* be an optimal policy, which maximizes the return from any start state. From [6], it is well known that there exists an optimal strategy, which is a deterministic policy. This implies that for any policy π and any state s, we have $V^{\pi^*}_M(s) \geq V^\pi_M(s)$ and $\pi^* = argmax_a(R_M(s,a) + \gamma(\sum_{s'} P^M_{s,s'}(a) Q(s',b)))$. The optimal policy is also the only fixed point of the operator,

$$(TQ)(s,a) = R_M(s,a) + \gamma(\sum_{s'} P^M_{s,s'}(a) Q(s',b)))$$

3. Q-learning

The Q-learning algorithm estimates the state-action value function (for discounted return) as follows:

$$Q_{t+1}(s,a) = (1-\alpha_t(s,a)) Q_t(s,a) + \alpha_t(s,a)(R_M(s,a) + \gamma \max_{b \in U(s')} Q_t(s',b)) \tag{3}$$

where s' is the state reached from state s when performing action a at time t. Let $T^{s,a}$ be the set of times, where action a was performed at state s, then $\alpha_t(s,a) = 0 \, \forall t \notin T^{s,a}$. It is known that (Refer [1]) Q-learning converges to Q^* if each state action pair is performed infinitely often and $\alpha_t(s,a)$ satisfies for each (s,a) pair:

$$\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty \text{ and } \sum_{t=1}^{\infty} \alpha_t(s,a)^2 < \infty$$

Q-learning is an asynchronous process in the sense that it updates a single entry each step. We describe synchronous and asynchronous algorithms below:

- Synchronous Q-learning algorithm :
  - $\forall\, s,a : Q_0(s,a) = C$
  - $\forall\, s,a : Q_{t+1}(s,a) = (1-\alpha^\omega_t) Q_t(s,a) + \alpha^\omega_t (R_M(s,a) + \gamma \max_{b \in U(\overline{s})} Q_t(\overline{s},b)$

  where $\overline{s}$ is the state reached from state s when performing action a and C is some constant. The learning rate is $\alpha^\omega_t = \frac{1}{(t+1)^\omega}$ for $\omega \in (\frac{1}{2}, 1]$
- Asynchronous Q-learning algorithm : This is simply regular Q-learning algorithm as defined in 1 with the following updates:
  - $\forall\, s,a : Q_0(s,a) = C$
  - $\forall\, s,a : Q_{t+1}(s,a) = (1-\alpha^\omega_t(s,a)) Q_t(s,a) + \alpha^\omega_t(s,a)(R_M(s,a) + \gamma \max_{b \in U(\overline{s})} Q_t(\overline{s},b)$

where $\overline{s}$ is the state reached from state s when performing action a and C is some constant. Let $\#(s,a,t)$ be one plus the number of times, until time t, that we have visited state s and performed action a. The learning rate $\alpha_t^\omega(s,a) = \frac{1}{[\#(s,a,t)]^\omega}$, if $t \in T^{s,a}$ and $\alpha_t^\omega(s,a) = 0$, otherwise.

Reinforcement learning technique is explored to reduce slippages in quantitative/high frequency trading in Indian stock futures by utilizing underlying market microstructure variables.

When a volume to trade and fixed time horizon to trade this volume are given, in general hostorical volume profile will be adopted so that overall cost of trading will be improved. This sort of algorithms are tradititionally referred as execution algorithms or sell side algorithms. Some standard execution algorithms are Volume weighted average price (VWAP), Time weighted average price (TWAP) & Almgren-Chriss model.

These traditional models have limitations applying to quantitative trading. In recent studies, [3] and [5] applied reinforcement learning for optimal liquidation problems. In the present study, we apply derived a modified version of Q-learning and apply to quantitative trading and test it using Indian Futures.

Training reinforcement learning agent by using market microstrucutre variables which capture spread, order book imblance and traded volume(both historical and extrapolated), we could improve trading cost 10% compared with VWAP, TWAP, Kearns and Hendricks models. We also compare with submit-and-leave and market order policies as in [3]

Improvement results mentioned here are just indicative. We will modify them at the end

One of the challenging problems faced by traders & portfolio managers in high frequency trading is optimal execution of their orders in order to reduce slippages. Most of the times, underlying trading strategies would give good theoretical results but these strategies fail to deliver similar results in production due to high slippages. Here the slippages can be interpreted as the price difference between reference price and executed price, i.e., adverse deviations of actual transaction prices from an arrival price baseline when the trading decision is made.

Define slippage?

Traders also measure slippage as a devition from the market volume-weighted trade price (VWAP) or time-weighted trade price (TWAP) over the trading period. Here trader compares his performance with that of the market. The main issue faced by trader is the compromise between price impact and opportunity cost when executing an order.

Q-learning is a model-free reinforcement learning technique. Q-learning works by learning an action-value function that gives us the expected value of taking a given action in a given state. After this action-value function is learned, our agent follows this policy for a maximal expected reward.

By using the order book dynamics and market microstructure, we explore reinforcement learning to improve present optimal order execution strategies. For a given trade decision(buy or sell), fixed time(eg: next 10 minutes) and volume to trade(eg: 10 lots), we aim to find a optimal set of actions(buy by standing at different level of order book) which is dynamic with respect to favourable/ unfavourable conditions there by minimizing the overall cost of trading.

We take submit and leave policy as a base model.

By training the model to modify actions based on market spread, volume imbalance, trend and traded volume, we are able to improve post-trade implementation shortfall by up to 60% on average compared to the based model and 20% on average compared with RL model proposed in [3]. based on 150+ stocks and sample volumes to trade.

4. Reinforcement learning algorithms

4.1. **Online algorithms.**

4.2. **Offline algorithms.**

5. Q learning algorithm

Q-learning is an off-policy algorithm. It can be proven that given sufficient training, the algorithm converges with probability 1 to a close approximation of the action-value function for an arbitrary target policy.

Initialize Q(state, action) arbitrarly
for *each episode* **do**
  Initialize state
  for *each step of episode* **do**
    Choose **action** from **state** using policy derived from Q
    Take action **action**, observe r, state'
    $Q(state, action) = Q(state, action) + \alpha[r +$
     $\gamma \max_{action'} Q(state', action') - Q(state, action)]$
    state = state'
  end
  Until state is terminal
end

Algorithm 1: States preparation

5.1. **SARSA.**

6. Kearns ([3]) and Hendricks ([5]) algorithms

Kearns algorithm:

```
OptimalExecution(V, H, T, I, L)
for t = T to 0 do
    while not end of data do
        Transform(order book) → o_1, ··· , o_R
        for i = 0 to I do
            for a = 0 to L do
                Set x = {t, i, o_1, ··· , o_R}
                Simulate transition x → y
                Calculate c_im(x, a)
                Look up max_p c(y, p)
                Update c(< t, i, o_1, ··· , o_R, a >)
            end
        end
    end
end
Select the highest-payout action max_p c(y, p) in every state y to output
 optimal policy
```

Algorithm 2: Kearns algorithm

```
Hendricks algorithm:
OptimalExecution(V, T, I, A)
for Episode 1 to N do
    Record reference price at time t = 0
    for t = T to 1 do
        for i = 1 to I do
            Calculate episode's STATE attributes < s, v >
            for a = 1 to A do
                Set x =< t, i, s, v >
                Determine the action volume a
                Calculate IS from trade, R(x,a)
                Simulate transition x to y
                Look up max_p Q(y, p)
                Update Q(x,a) = Q(x,a) + alpha*U
            end
        end
    end
end
Select the lowest IS action max_p Q(y, p) for optimal policy
```

Algorithm 3: Hendricks algorithm

7. Market analytics

1 lot future value is approximately 0.5 million rupees. Irrespective of price range, all stocks, futures and options will have tick size of 5 paise. Exchange trading costs in market: 1 lot future buy/sell Exchange trading costs in market: 1 lot option Exchange trading costs in market: cash buy or sell

- What is the trade percentage from top of the order book updates?

- Present results from Nifty, Bank Nifty, top ten liquid stocks, high price stocks and low price stocks.
- Filter out stocks based on price thresholds.
- Present a case to stand with a new level created than taking existing level from market
- Present number of trades with 1/2/3/4/5 lots.
- Present number of orders in the top of the order book
- TBQ and TSQ analysis can be presented

We do the following data processing:

- Stock information is divided into 5 states: s1 to s5
- 10 seconds sampled stock data
- Time states = 5 minutes * 6 = 30
- 5 minutes starting state t1 and ending state t30
- Number of states(stock states + time states) = 5*30 = 150
- 150 states can be represented as (si, tj), i=1 to 5, j= 1 to 30
- (si, t30), i=1 to 5, are absorbing states
- Possible actions at each state: Buy with second best bid price BP2 or Buy with best bid price BP1 or Buy with mid price MP or buy with best ask price AP1 or buy with second best ask price AP2.
- Observe that BP2 is the lowest price and AP2 is the highest price. Observe that at BP2, BP1 and MP, we may not get any seller, thus we end up not buying the stock.
- We update our order price every 10 seconds once

Our aim to buy before 5 minutes ends (from the point we made decision to buy). We find optimal action for each state (si, tj) for i=1 to 5 and j = 1 to 30.

We will have the following initial Q value count table (QCT) and Q value table(QVT) tables:

$QCT =$

State & Action	BP2	BP1	Mid	AP1	AP2
(s1, t1) (initial state)	0	0	0	0	0
(s2, t1) (initial state)	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮
(s5, t30) (absorbing state)	0	0	0	0	0

$QVT =$

State & Action	BP2	BP1	Mid	AP1	AP2
(s1, t1) (initial state)	0	0	0	0	0
(s2, t1) (initial state)	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮
(s5, t30) (absorbing state)	0	0	0	0	0

Suppose our target is buy. The logic presented below will hold true for sell as well with appropriate changes in slippage and reward calculations.

7.1. **States preparation.** Consider 10 seconds sampled order book data and consider sizeRatio. For the trainData, get sizeRatio data and call it as sizeRatioArray (sizeRatio will be one of the columns in trainData). Calculate the following four quantities:

- srTh1 = quantile(sizeRatioArray, 20%)
- srTh2 = quantile(sizeRatioArray, 40%)
- srTh3 = quantile(sizeRatioArray, 60%)
- srTh4 = quantile(sizeRatioArray, 80%)

Now prepare sizeRatio states as follows:

for *time_i in 1:length(sizeRatioArray)* **do**
  if *sizeRatioArray[time_i] < srTh1* **then**
    sizeRatioState = 1
  if *(sizeRatioArray[time_i] >= srTh1) & (sizeRatioArray[time_i] < srTh2)* **then**
    sizeRatioState = 2
  if *(sizeRatioArray[time_i] >= srTh2) & (sizeRatioArray[time_i] < srTh3)* **then**
    sizeRatioState = 3
  if *(sizeRatioArray[time_i] >= srTh3) & (sizeRatioArray[time_i] < srTh4)* **then**
    sizeRatioState = 4
  if *sizeRatioArray[time_i] >= srTh4* **then**
    sizeRatioState = 5
end

Algorithm 4: States preparation

These thresholds, srTh1, srTh2, srTh3, srTh4 should be used to make states in production as well. This procedure will be same for any other feature.

8. Updating QcountTable and QvalueTable

Suppose our start time is at 10:30:00. Consider data between 10:30:00 and 10:35:00 (5 minutes data). We will have totally 30 data points between 10:30:00 and 10:35:00. We start at 10:34:50 and consider 10 seconds sampled data(at 10:34:50) and trade data between 10:34:50 and 10:35:00. At this stage our remaining time is 1. So our time state is t30. Now we look at market state number at 10:34:50. Suppose this is s4. Then our state is (s4, t30). Get the last traded price from previous 5 minutes bucket, i.e., 10:25:00 to 10:30:00. Call this price as refPrice.

Now we run through all 5 actions. Suppose we start with bid2 (BP2). Now look at 10 seconds sample data at 10:34:50 and its BID_PRICE2. Since our target is buy, we consider ASK_PRICE1 though our action is bid2. Now slippage is calculated as

$$slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$$

while $presentTime > 10:34:50$ *and* $presentTime < 10:35:00$ **do**
  if $action = bid2$ **then**
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
  else if $action = bid1$ **then**
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
  else if $action = mid$ **then**
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
  else if $action = ask1$ **then**
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
  else if $action = ask2$ **then**
    $slippage = \frac{(refPrice - ASK\_PRICE2)}{refPrice} * 10000$
end

Algorithm 5: Absorbing state slipplage calculation

Now, we update QcountTable as follows:

$$QCT[(s4, t30), bid2] = QCT[(s4, t30), bid2] + 1$$

For updating QVT, we use the following table:

while $presentTime > 10:34:50$ *and* $presentTime < 10:35:00$ **do**
  if $(action = bid2)$ **then**
    $\alpha = \frac{1}{QCT[(s4,t30),action]}$
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
    $QVT[(s4, t30), action] =$
     $(1-\alpha) * QVT[(s4, t30), action] + \alpha * (slippage + 0.75 * slippagePenalty)$
  else if $(action = bid1)$ **then**
    $\alpha = \frac{1}{QCT[(s4,t30),action]}$
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
    $QVT[(s4, t30), action] =$
     $(1-\alpha) * QVT[(s4, t30), action] + \alpha * (slippage + 0.50 * slippagePenalty)$
  else if $(action = mid)$ **then**
    $\alpha = \frac{1}{QCT[(s4,t30),action]}$
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
    $QVT[(s4, t30), action] =$
     $(1-\alpha) * QVT[(s4, t30), action] + \alpha * (slippage + 0.25 * slippagePenalty)$
  else if $(action = ask1)$ **then**
    $\alpha = \frac{1}{QCT[(s4,t30),action]}$
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
    $QVT[(s4, t30), action] = (1-\alpha) * QVT[(s4, t30), action] + \alpha * (slippage)$
  else if $(action = ask2)$ **then**
    $\alpha = \frac{1}{QCT[(s4,t30),action]}$
    $slippage = \frac{(refPrice - ASK\_PRICE1)}{refPrice} * 10000$
    $QVT[(s4, t30), action] = (1-\alpha) * QVT[(s4, t30), action] + \alpha * (slippage)$
end

Algorithm 6: Absorbing state QvalueTable updation

Suppose we consider data from 10:34:40 to 10:34:50. Here, remaining time is 2. So our time state is t29. Suppose our market state s2. Then our present state is (s2, t29). Suppose our action is bid1 and our price is BID_PRICE1. In the next 10 seconds, we may get fill or we may not get fill. We now calculate next state which is (s4, t30). Let us say presentState = (s2, t29) and nextState = (s4, t30)

Now QVT is updated as follows:

while $presentTime > 10:34:40$ *and* $presentTime < 10:34:50$ **do**
  if $(action = bid2)$ **then**
    $QCT[presentState, action] = QCT[presentState, action] + 1$
    $\alpha = \frac{1}{QCT[(s4,t30),action]}$
    if *BP2 > min trade prices between 10:34:40 and 10:34:50* **then**
      $slippage = \frac{(refPrice - BP2)}{refPrice} * 10000$
      $QVT[presentState, action] =$
      $(1 - \alpha) * QVT[presentState, action] + \alpha * (slippage)$
    else
      $slippage =$
      $notFillTimePenalty * log(remainingTime/totalTimePoints)$
      $tempSlippage = \frac{(refPrice - BP2)}{refPrice} * 10000$
      if $tempSlippage >= 0$ **then**
        $QVT[presentState, action] =$
        $(1 - \alpha) * QVT[presentState, action] + \alpha * (slippage +$
        $\max(QVT[nextState,]))$
      else
        $slippage = slippage + tempSlippage$
        $QVT[presentState, action] =$
        $(1 - \alpha) * QVT[presentState, action] + \alpha * (slippage +$
        $\max(QVT[nextState,]))$
end

Algorithm 7: Non-absorbing state QvalueTable updation

9. State aggregation

Features for state aggregation:

- Remining inventory
- Elapsed time
- Short term trend
- Volatility
- Order book imbalance
- Bid-ask spread etc.

10. RL algorithm workflow is as follows

Step1: Get 10 seconds sampling orderBook data with sizeRatio, ofi, imbalance, L1 spread, trend, and L5 price data

Step2: Get trade and L1 data

Step3: Prepare states for the selected market features for the train data(Ref: Section:)

Step4: Prepare states for the selected market features for the test data using train data thresholds
Step5: Combining multiple features, refer to Section:
Step6: Update Qvalue and QvalueCount tables
Step7: In the given timeEpisode, run RL algorithm on train data (Ref: Section)
Step8: Repeat Step5 for given day and for the entire train period
Step9: Find optimal policy
Step10: Run test data with the optimal policy
Step11: Generate metrics

11. Optimal policy

For all the states we have, go through all the actions (bid5 to ask5, in total 11 actions) and update the expected cost associated with that action (taking notFillTimePenalty into consideration) and following optimal policy afterwards. Due to our action, if trade happens, we move to next private volume remaining state. This private state along with private time remaining and market states give us new state for the next time episode. Since reinforcement learning moves backwards in time, we have already optimized this new state with the expected cost of following the optimal strategy from that state. Our cost updation equation is as follows:

$$c(x,a) = \frac{n}{n+1}c(x,a) + \frac{1}{n+1}[c_{im}(x,a) + \max_{p} c(y,p)] \tag{4}$$

where

- $c(x,a)$: the cost of taking action a in the state x and following the optimal strategy afterwards
- $c_{im}(x,a)$: the immediate cost of taking action a in the state x
- y: the new state where we end up after taking action a in x
- n: the number of times we have tried a in x
- p: action taken in y

12. Optimal execution

13. Optimal execution in live

Two inputs are required to be precalculated:

- State-Action table
- Trend Parameter Percentile Values (Will be calculated based on Train Data)

The following input variables are required:

- Parameter time: OFI/Micro/Imblance
- Time Window1: 5 minutes
- Time Window2: 10 seconds
- OFIWindow: 120 seconds
- SizeRatioWindow: 300 seconds
- StopLoss: 30 basis points
- Alpha: Dynamic
- NotFillTimePenalty: 0 (can check this condition)

13.1. **Trend parameters are calcualted as follows.**

- Imbalance: Consider Ticks only when there has been an update in at least one level of the order book (Out of Top 5 Levels). Any change is price or size is considered an update

$$Imbalance = \frac{TBQ}{TSQ}$$

- Micro5L: Consider Ticks only when there has been an update in at least one level of the order book (Out of Top 5 Levels). Any change is price or size is considered an update.

$$MicroPrice = \frac{\sum_{i=1}^{5}(AskPrice_i * BidSize_i + BidPrice_i * AskSize_i)}{\sum_{i=1}^{5}(BidSize_i + AskSize_i)}$$

Production implementation is as follows:

- The trend feature value will be calculated at every 10 second interval (Time Window2)
- The trend feature we would like to use for the RL Algo classification will be dependent on the ParameterType as defined by the user.
- The Order Placement/Modification will begin once the execution engine receives a signal from the strategy engine.
- As soon as signal is received, the feature value is calculated and mapped to its respective percentile bucket from the 'Trend Parameter Percentile Values' file to determine the market state
- The Maximum time limit allowed for a fill is 5 minutes (Time Window1). If Time since Signal is greater than TimeWindow1, then force a fill by hitting market.
- Time Remaining till TimeWindow1 will determine the Private State. Look up the optimal action from 'State-Action Table' file for corresponding Market and Private State
- If order is not filled in current 10 second window, recalculate feature value and repeat step 6
- Stop Loss functionality will also be present with value of 30 basis points (StopLoss: UserDefined)
- No Profit Book to be implemented

References

[1] Dimitri P. Bertsekas and John N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

[2] Eyal Even-Dar and Yishay Mansour, *Learning Rates for Q-learning*, Journal of Machine Learning Research 5 (2003) 1-25.

[3] Y. Nevmyvaka, Y. Feng and M. Kearns, *Reinforcement learning for optimized trade execution*, Proceedings of the 23rd international conference on machine learning, pp. 673-680, 2006.

[4] T. Chakraborty and M. Kearns, *Market Making and Mean Reversion*, 2011.

[5] Dieter Hendricks and Diane Wilcox, *A reinforcement learning extension to the Almgren-Chriss framework for optimal trade execution*, 2014.

[6] Martin L. Puterman, *Markov Decision Processes—Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., New York, NY, 1994.

[7] Csaba Szepesvari, *Algorithms for Reinforcement Learning*, Synthesis Lectures on Arti cial Intelligence and Machine Learnin, 2009.

[8] R. Sutton and A. Barto, *Reinforcement Learning* , MIT Press., Cambridge, MA., 1998.

[9] Chaiyakorn Yingsaeree, *Algorithmic trading: Model of execution probability and order placement strategy*, PhD Thesis, University College London, 2012.
[10] *http://hallvardnydal.github.io/new_posts/2015-07-21-deep_q/*, weblink.