Optimization of High-Frequency Trading Systems

David Sweet
dsweet@andamooka.org

- ex, trading strategy: buy/sell for profit
- ex, execution system: fill an order for a customer
High-Frequency Trading

- Dynamics/signals on order of seconds
- Demanding computer, network, & software engineering
- Models, Statistics, Machine Learning
- Revenue-generating, agency execution

- ongoing “computerization” of trading — like every other industry
- “program trading” in 1980s
- “electronic/algorithmic trading” in 1990s & early 2000s
- “high-frequency trading” since then
- network includes telecom; microwave, millimeter wave
High-Frequency Trading

- Arbitrage: inter-asset, inter-exchange
- Liquidity Provision: passive, active
- Execution: reduce costs for customer

- all three strategies in existence “forever”
- started manually, improved with telecom, computers, algorithms
- still improving
HFT: Where is the value?

- HFT strategies make money because they make markets more efficient and market participants are willing to pay for that

- arbitrage makes sure assets are priced fairly ("price discovery"), so you get a fair price when you trade
- liquidity provision reduces the spread (the cost of trading on an exchange)
- execution algos reduce costs of executing an order (spread, fees, market impact)
- HFT is a red herring; if everyone only traded once/day (i.e. extremely slowly) these strategies would still lower costs
A Trading Strategy

If signal > threshold then Buy
If -signal > threshold then Sell
If end of day, liquidate and stop

Rule set called a policy
threshold is a parameter

- Keep this example in mind as we go along
- policy answers, “What should I do now?”
- Best threshold value depends on cost to trade, signal quality, how fast signal changes (decorrelates), cost to liquidate at EOD, and your definition of strategy quality (pnl, pnl - risk, etc.)
- How do you find the best threshold? That’s the subject of this talk…
Optimization

• Measure *quality* of parameters by trading
• Measurement Costs: losses, risk, opportunity
• Goal 1: Find highest-quality parameters
• Goal 2: Minimize cost of measurement

- quality: pnl, pnl - risk
- Every day that you trade at a suboptimal parameter — even if you’re making money — you’re paying an opportunity cost. You’ve missed out on the extra money you would have made by trading at a better parameter setting.
- competing goals
• Simulation is cheap
• But: Market reacts to our actions
• But: Hidden liquidity is … hidden
• But: Latencies complicated
• Simulation not useful for parameter optimization

- easy to run hundreds or thousands of simulations to test different parameters
- matching engine processes our orders — even if they don’t get filled; takes time, changes market
- other traders (computers) see our orders/executions in public data and make different decisions than they would/could have
- Any visible queue can have hidden liquidity, too + dark pools = more hidden queues than visible; *most* queues are hidden
- 24% of US Equities traded volume dark/hidden [Rosenblatt’s Monthly Dark Liquidity Tracker, December 2016]
- longer-term strategies can treat all of these effects as a small, noisy cost; but they are significant for HFT where profits/share are on par with these costs
- latencies possible at every network node; latencies correlated to each other and likely to your signals
- simulation still useful for testing code quality, studying optimization methodology, estimating some operational risks
A/B Test

- Compare two strategies (policies)
- Call them “Policy A” and “Policy B”
- Ex: threshold=1 vs. threshold=2
- Ex: “Trade through JPM” vs. “Trade through GS”

- can compare real-valued parameter values or categorical, non-parameterized design decisions
A/B Test

- Trade A and B side-by-side for N days
- N determined by noise level and desired precision

- Ask, “Is B better than A?”
Improving A/B

- Lower cost of measurements
- Compare more possibilities

- What if B is a *lot* better? Can’t we stop early and lower the cost? [No, b/c your plan to deal with noise required N days.]
- What if we have more than two options to compare? A, B, C, ...? A vs. B, then winner vs. C, then ... This could take a long time (and be very expensive).
A/B Test

- Design of Experiments
- Response Surface Methodology
- Policy Search
- Actor-Critic

Multi-Armed Bandit

- Contextual Bandit
- Q-Learning

Policy Search

- Q-Learning
Design of Experiments

• Evaluate multiple parameters’ settings

• Choose which parameter values to measure to keep information high and cost low

- ex: threshold = 1, 2, 3, ...
- *not* JPM vs GS, however
- Factorial: all combinations, $2^n$ measurements
- Fraction Factorial: Try to assess each parameter independently by removing pair-wise correlation;
- avoid: “Hey! When I increased p1, quality improved!” “But when you increased p1 you also increased p2. So which parameters is responsible for the improvement?”
- Fewer measurements = lower cost
D.O.E. adds support for multiple parameter (MULTI) values and consideration of measurement cost (COST)
Response Surface Methodology

- Build a model (regression) from data collected via D.O.E.
- *Infer* the best parameters from model!
- Verify/Improve: D.O.E. around inferred-best

- The “best” parameters likely won’t be in the data set.
- Re-center the measurements around the inferred-best. Then take measurements to verify your inference.
- Repeat if desired until your inferred-best stops changing.
- This is an iterative (manual) optimization routine
RSM adds data efficiency by modeling quality vs. parameters (DEP).
Policy Search

- Automates RSM
  1) Model response surface
  2) Find inferred-best parameters
  3) Design next experiment
  4) Go to (1)

- many algorithms; good search terms: Bayesian Optimization, Efficient Global Optimization, Black-Box optimization with “expensive” objective functions
- (3) tries to optimally trade off the need to collect more data (to build a better model) which has a cost and the desire to trade at the optimal parameters; aka “exploration vs. exploitation”
- exploitation => higher revenue now; exploration => higher revenue in the future
- accounts for noise / uncertainty in each measurement, so each trading day can use a new experiment design; all data are combined optimally into RSM
- Policy Search adds a method to optimally design the next experiment (EXP).
- Includes “exploitation vs. exploration” trade-off when designing next experiment.
- Optimization & trading are now one continuous, on-going process. Compare this to A/B testing where there are two “phases”: run the experiment to learn what’s best, then use that information to trade.
- pay some measurement cost today for higher quality tomorrow
• “one-armed bandit” == slot machine
• MAB: K arms, each with different, noisy payout
• Strategy to optimize total payout?

- MAB is a problem definition
- “MAB methods” are ways to solve that problem
- K=2 arms == a more efficient A/B test
- MAB cares about measurement cost
- MAB handles multiple choices: could be different parameter values (threshold=1,2,3) like DOE, but could also be qualitatively different choices (compare code revisions, hardware, order types, brokers)
Multi-Armed Bandit Methods

1. Pull each arm several times
   \[ Q(\text{arm}) = \text{mean(arm quality measurements)} \]
   Thereafter only pull highest-Q arm

2. \( p=.9 \): pull highest-Q arm
   \( p=.1 \): pull random arm

3. Pull arm with highest \( Q + \text{stddev(quality)} \)

- (1) spends a lot of time measuring, but ultimately pulls the best
- (2) “explores” 10% of time to improve estimates, but usually (90% of time) pulls the one we think is best; but never stops exploring
- (3) expression makes exploration vs. exploitation explicit; adds more samples to the noisier estimates (more efficient exploration); eventually stops exploring (more efficient exploitation)
- MAB measures multiple options (MULTI)
- MAB is sensitive to cost of measurements (COST)
- MAB “designs” series of experiments (EXP)
- Compare to DOE:
  - DOE compares real-valued parameter values, designs one big, low-noise measurement
  - MAB compares arbitrarily-defined options, “designs” a series of small, noisy measurements
Contextual Bandit

- context (aka. state) == signals, time of day, product traded, etc.
- $Q(\text{arm, context}) = \text{regression model}$
- Fit model from measurements so far

- Follow same rules as MAB — 90%/10% or maximal mean+std, except means are replaced by conditional means, i.e. model's prediction of arm quality
- Contextual Bandit increases data efficiency by modeling arm quality vs. state (DES) and quality vs. arm (DEP)
- Notice shift in mindset: “arms” now intraday decisions instead of just candidates for where to fix parameters
Q-Learning

- What if “arms” were buy, sell, wait?
- Consecutive arm pulls not independent
- \( Q(t, \text{arm}, \text{context}) = \text{qualityMeasurement}(t) + \text{qualityMeasurement}(t+1) + \text{qualityMeasurement}(t+2) + \ldots \)
- \( Q \) function determines the policy

- arm pulls were independent in MAB and Contextual MAB
- \( Q \) estimate now depends on future contexts *and* your future decisions
- fitting methods can be complex; won’t cover here
- \( Q \) determines whole trading strategy: Which arm has highest \( Q \)? (or highest \( Q + \text{stddev}Q \), to include exploration)
- Q-Learning models sequences of decisions (SEQ) that are not independent
- Methods optimize policy parameters
- Don't consider what policy is doing, just look at your measurement of quality
- Very flexible: Your strategy (policy) can be designed any way you like.
- Data is used efficiently by modeling quality vs. parameters.
- Generally works only with a small number of parameters (~5).
Methods look at policy step by step

Actions are discrete

Methods analyze actions individually

- To use Contextual Bandit or Q-Learning you need to write your strategy (policy) in a compatible way: You need a context (signals) and arms (buy, sell, hold).
- In return you get very efficient use of data through models of quality vs context.
Actor-Critic

- Combines Policy Search and Q-Learning
- Allows black-box policy
- More policy parameters
- Most efficient use of data

- data efficiency comes from modeling Q vs. context and policy parameters
- optimize policy parameters using model Q(arm, context, parameters) as objective instead of simpler model of Q(parameters)
Themes, going from top to bottom:
- increasing number of arms/parameters
- increasing data efficiency: build more sophisticated models of the data collected so far
- increasing exploration efficiency
Continuous Optimization

- Tune to real markets’ complex dynamics
- Try more ideas, more quickly & efficiently
- Adapt to changing markets

Practical way to view strategy design: as a continuous, never-ending optimization