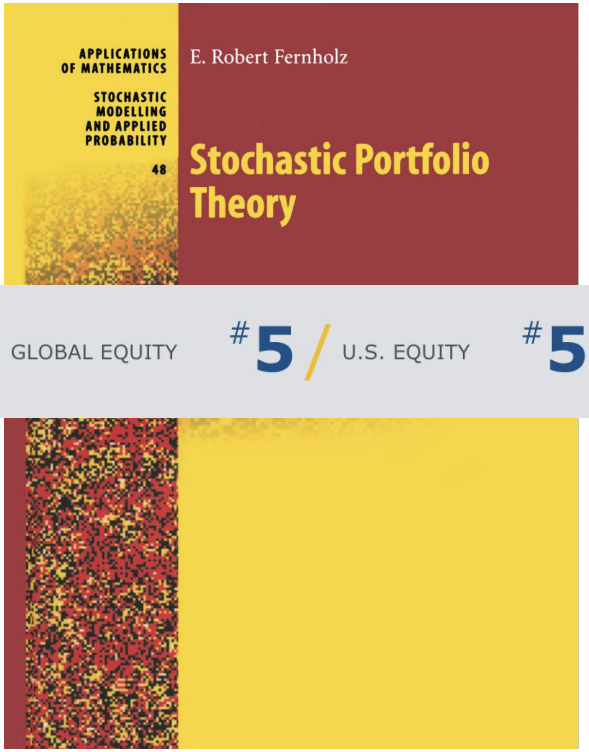
Peter Cotton
Chief Data Scientist
Intech Investments

The Traveling Algorithm Problem

Princeton Quant Trading Conference
Sep 26, 2020

Hello. I work for ...

Intech Investments



RANKED AS A LEADING QUANT MANAGER BY AUM[1]

#4 / GLOBAL EQUITY

#5 / U.S. EQUITY

#5 / ENHANCED EQUITY

#6 / DEFENSIVE EQUITY

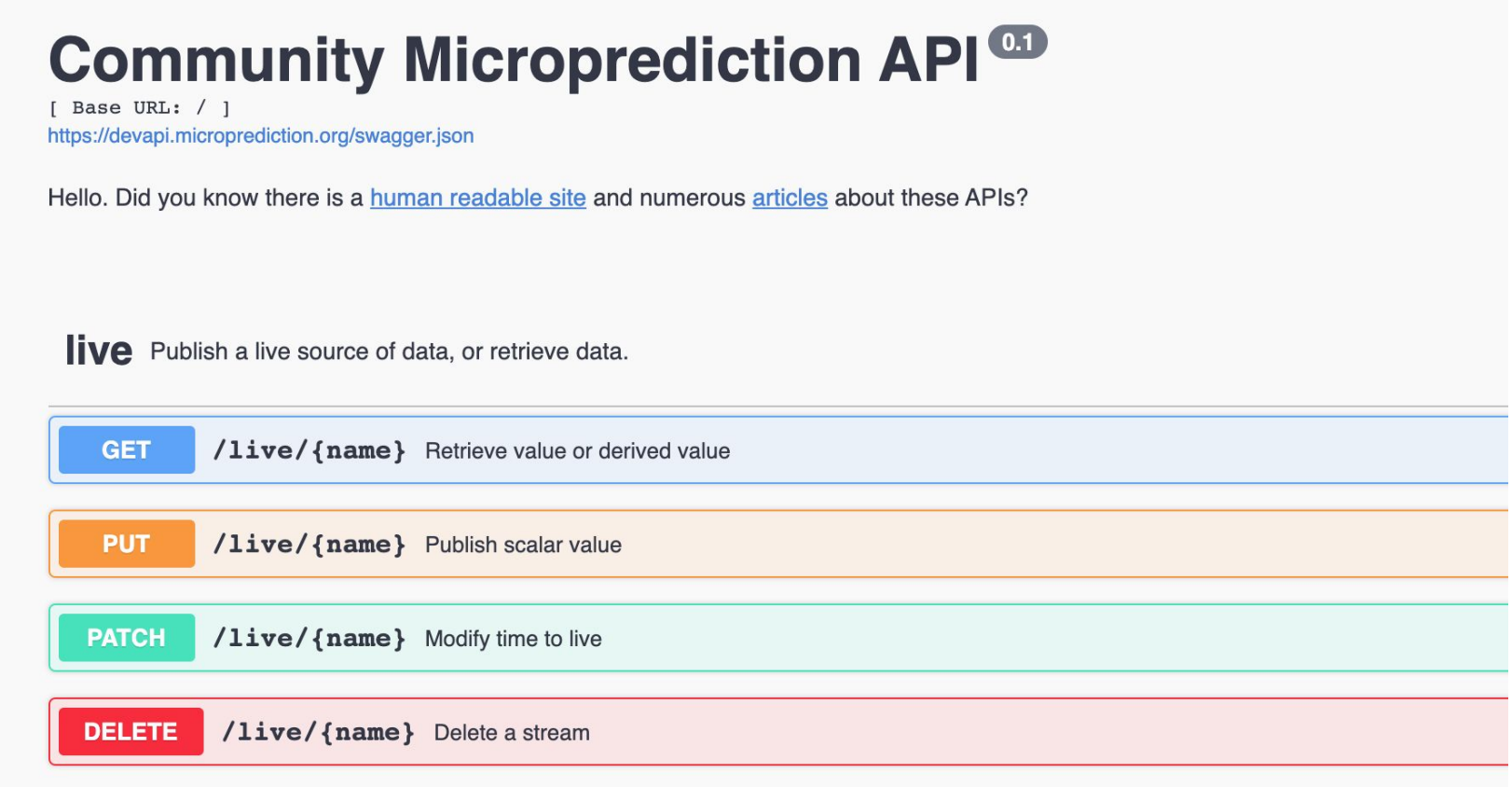
http://finmathblog.blogspot.com/2012/12/excess-return-on-portfolio-of-lognormal.html

There are two travelling algorithm problems

Problem #1: They don't.

Problem #2: A mathematical problem inspired by a proposed solution to Problem 1.

So here it is


Community Microprediction API
0.1
[Base URL: /]
https://devapi.microprediction.org/swagger.json
Hello. Did you know there is a human readable site and numerous articles about these APIs?
live
Publish a live source of data, or retrieve data.
GET
/live/{name}
Retrieve value or derived value
PUT
/live/{name}
Publish scalar value
PATCH
/live/{name}
Modify time to live
DELETE
/live/{name}
Delete a stream

Microprediction – What is it?

Constantly repeated prediction

Microprediction

- How to improve demand forecasting? (maybe but …hmmm…)
- How to improve short demand forecasting? (yes … better!)
- How to predict call center volumes? (yes)
- How to predict software failure? (yes, maybe, but …)
- How to forecast stock price? (yes, everything but the mean!)
- How to predict value functions (RL)? (yes, why not?)
- How to predict electricity prices? (yes)
- How to predict bakery sales? (yes)
- How to predict activity near New York city hospitals? (yes)
- How to predict the winner of an election? (no, maybe very indirectly)

What can you do with a microprediction API?

Standardize data relative to competitive community forecasts

z1~fcx~70

Go to Stream →

Horizon: 70 sec | 310 sec | **910 sec** | 3555 sec

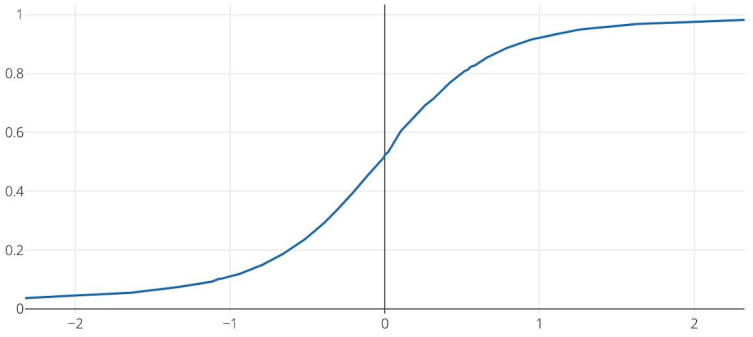
Leaderboard

Rank	MUID	Points
1	Comal Cheetah	+7.101
2	Decastyle Cat	+5.595
3	Exhalable Cat	+5.338
4	null	+0.8
5	Flammable Cod	-3.793
6	Cellose Bobcat	-15.04

Lagged Values

Timestamp	Z-Score
8/4 16:01:06	-0.51996
8/4 16:00:11	0.55508
8/4 15:59:18	-0.00448
8/4 15:58:05	-0.04721
8/4 15:57:04	1.48253
8/4 15:56:03	0.55323
8/4 15:55:05	-0.00446
8/4 15:54:03	-0.00445
8/4 15:53:04	0.58113
8/4 15:52:11	-1.29729
8/4 15:51:05	1.24723
8/4 15:50:06	-0.5483
8/4 15:49:04	-0.55082
8/4 15:48:04	-0.53286
8/4 15:47:04	-1.07452
8/4 15:46:06	-0.39177
8/4 15:45:13	0.26186

CDF



What can you do with a microprediction API?

Predict absolute values

traffic_absolute_speed

Live Current Value: **49.7**

← Go to Competitions | ← Go to Z1

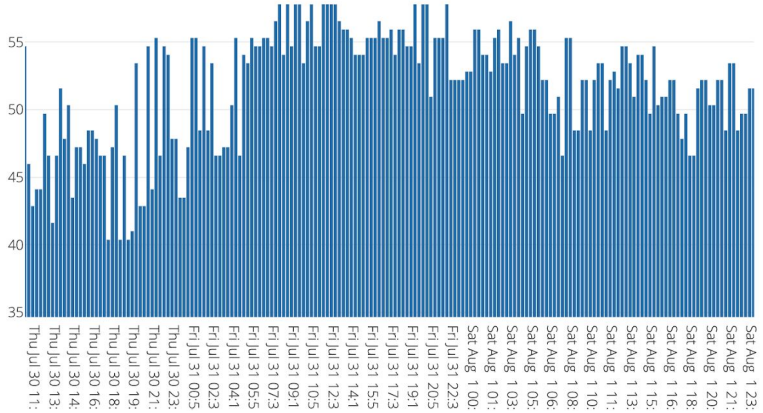
Leaderboard

Rank	MUID	Points
1	Staboy Bat	+231.03
2	Doodle Bee	+161.968
3	Chatty Fly	+101.565
4	Behest Mammal	+97.258
5	Decastyle Cat	-8.737
6	Lettable Clam	-8.816
7	Bedabble Toad	-9
8	Statesboy Cat	-10.129
9	Flammable Cod	-10.5

Lagged Values

Timestamp	Data
8/4 22:39:32	49.7
8/4 22:19:33	49.08
8/4 21:59:33	49.7
8/4 21:39:32	49.08
8/4 21:19:32	49.7
8/4 20:59:34	49.08
8/4 20:39:32	49.08
8/4 20:13:40	54.05
8/4 19:53:40	36.03
8/4 19:33:41	50.95
8/4 19:13:39	49.7
8/4 18:53:42	45.98
8/4 18:34:07	49.7
8/4 18:13:44	45.98
8/4 17:53:41	45.98
8/4 17:33:48	47.22
8/4 17:13:41	50.33

Data Values



What can you do with a microprediction API?

Predict changes in values

cop

Live Current Value: **1.0627**

← Go to Competitions | ← Go to Z1

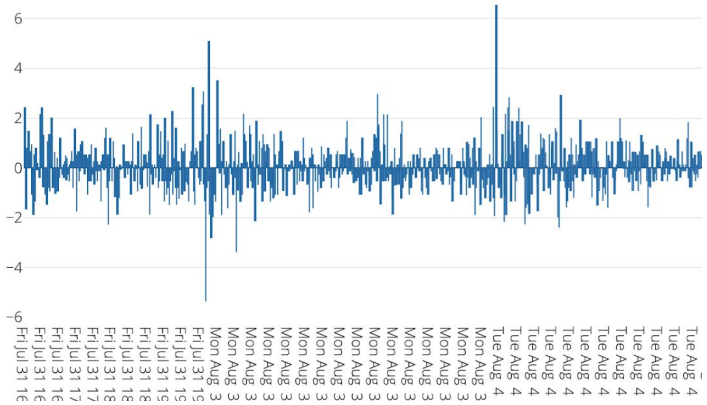
Leaderboard

Rank	MUID	Points
1	Cellose Bobcat	+44.149
2	Decastyle Cat	+31.662
3	Mesole Mammal	+11.565
4	null	+2.4
5	Bedabble Toad	-0.117
6	Bal Llama	-0.8
7	Lettable Clam	-3.291
8	Oft Sloth	-10.802
9	Exhalable Cat	-12.766

Lagged Values

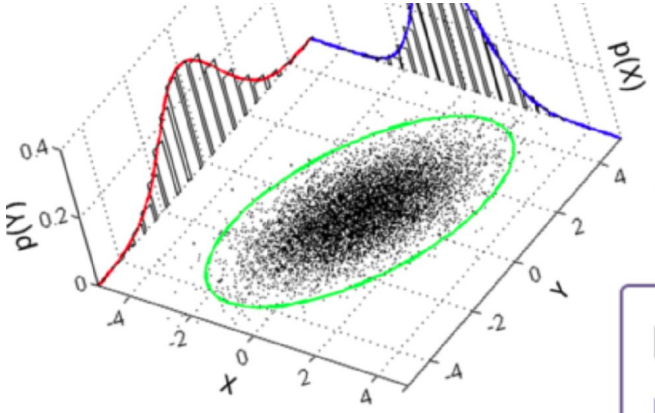
Timestamp	Data
8/4 16:01:05	1.0627
8/4 16:00:11	0.26585
8/4 15:59:18	0.53191
8/4 15:58:04	0.26606
8/4 15:57:03	0.13306
8/4 15:56:03	0.63094
8/4 15:55:04	-1.29605
8/4 15:54:03	1.86344
8/4 15:53:04	-0.66591
8/4 15:52:11	-1.19753
8/4 15:51:05	2.12993
8/4 15:50:06	-0.26649
8/4 15:49:04	-0.53277
8/4 15:48:04	-0.13315
8/4 15:47:04	-0.66547
8/4 15:46:06	-0.93091

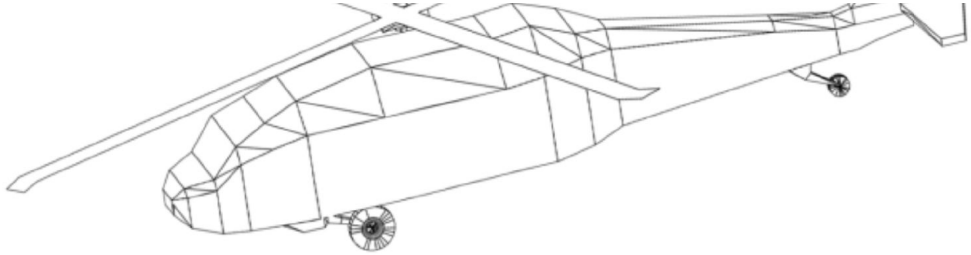
Data Values



What can you do with a microprediction API?

Crowdsource copulas (bivariate)





helicopter_theta

Live Current Value: **-0.67467**

← Go to Competitions | ← Go to Z1

Leaderboard

Rank	MUID	Points

Lagged Values

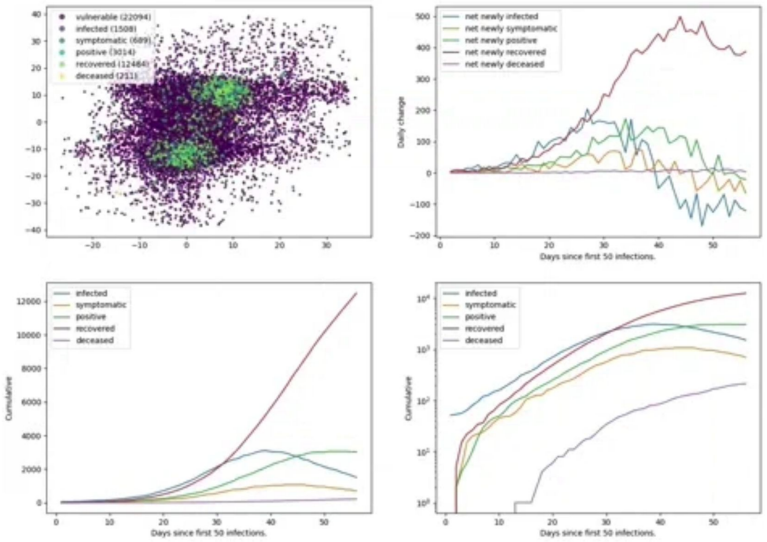
Timestamp	Data
7/8 08:54:09	-0.67467
7/8 08:47:09	-0.67589
7/8 08:40:09	-0.67386
7/8 08:33:08	-0.68115
7/8 08:26:08	-0.68034
7/8 08:22:12	-0.68155

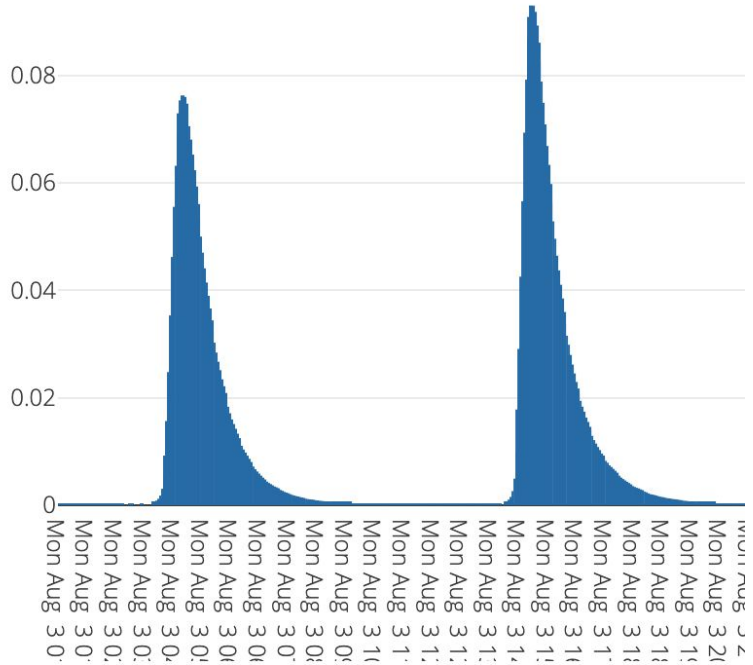
Explanation: https://www.linkedin.com/pulse/helicopulas-peter-cotton-phd/

What can you do with a microprediction API?

Surrogate models

https://www.microprediction.org/stream_dashboard.html?stream=pandemic_infected





Generative model: www.swarmprediction.com

Example: Predicting water levels for NOAA

```python
from credentials import ELFEST_BOBCAT  # You'll have to supply your own
import pandas as pd
from microprediction import MicroWriter
import time

# Video explanation of this example:
# https://vimeo.com/443203883

mw = MicroWriter(write_key=ELFEST_BOBCAT) # See creating_a_key.py
STREAM_NAME = 'water.json'


def water_height():
    df = pd.read_csv('https://www.ndbc.noaa.gov/data/realtime2/21413.dart')
    return float(df.iloc[1, :].values[0].split(' ')[-1])


def poll_for_an_hour():
    for _ in range(4):
        mw.set(name=STREAM_NAME, value=water_height())
        time.sleep(15 * 60)
        print('.', flush=True)


if __name__ == '__main__':
    poll_for_an_hour()
```

https://www.linkedin.com/feed/update/urn:li:activity:6694740397144596480/

How does it work?

You make it work!

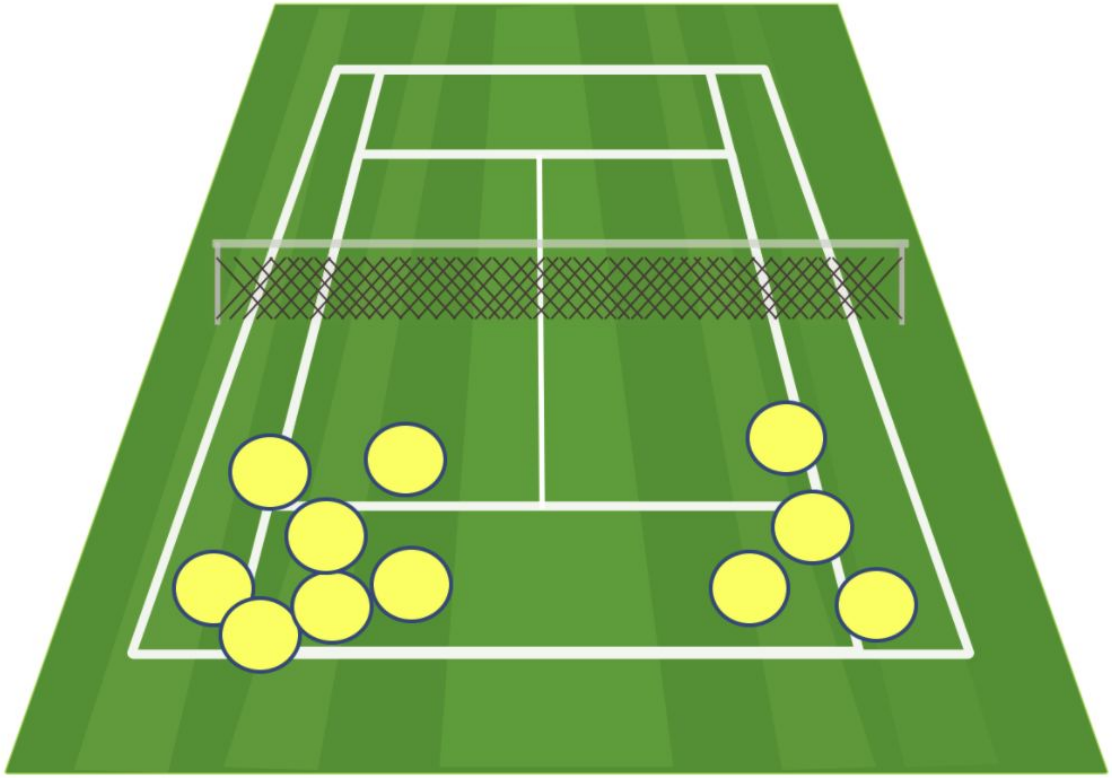
```
from microprediction import MicroCrawler

if __name__ == '__main__':
    crawler = MicroCrawler(difficulty=9)
    crawler.run()
```

Run this right now

70, 310, 910, 3555

Sends 225 guesses of the next point to arrive after ____ seconds have expired

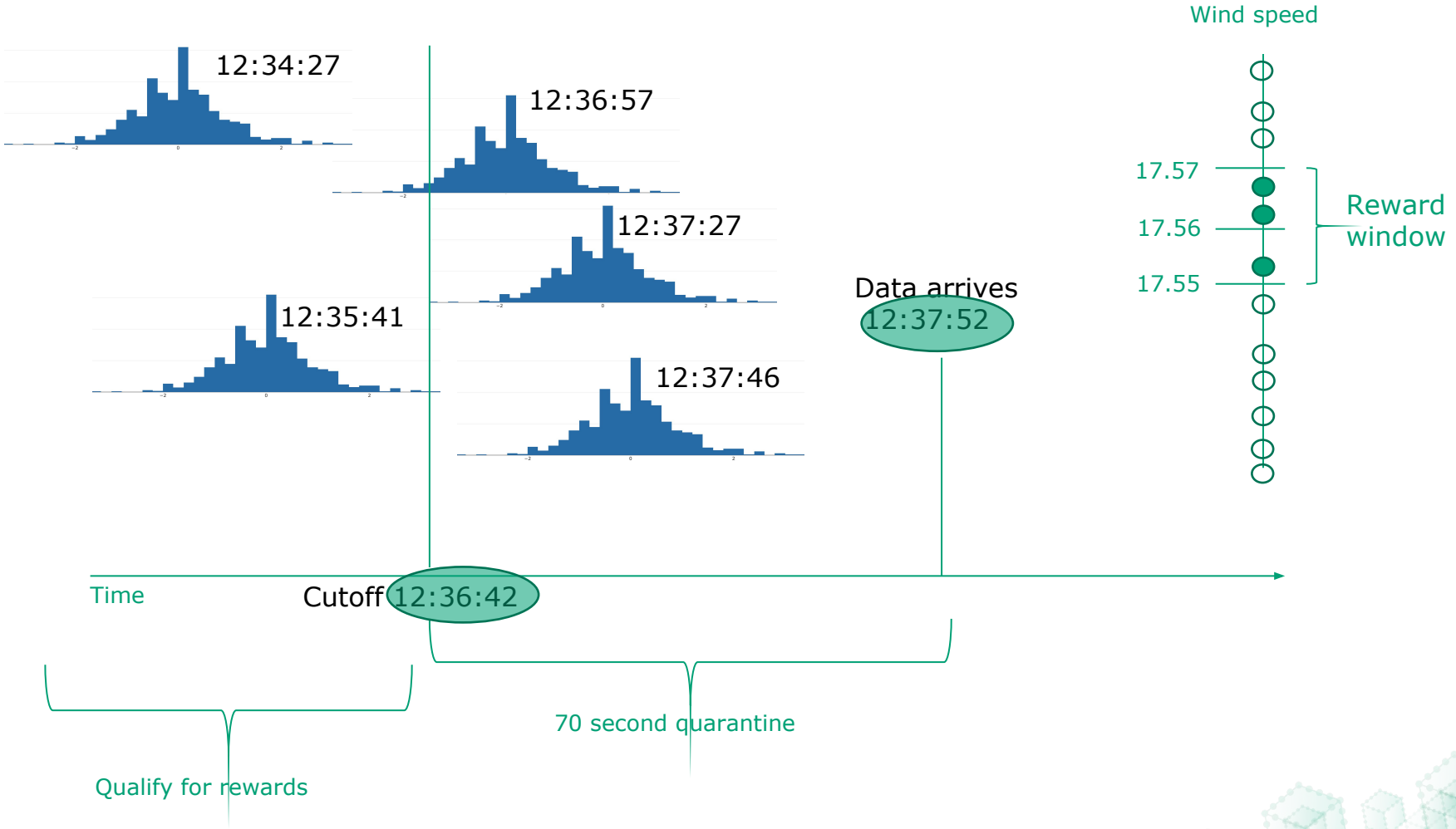
Why send 225 points instead of just the mean?



What can crawlers do?

1. Find exogenous data
2. Use automated model search
3. Use the prediction API recursively
4. Optimize their navigation
5. Predict when data will arrive
6. Exploit online statistical methods
7. Specialize to certain types of time series
8. Specialize to implied time series (zscores, zcurves)
9. Reset their performance
10. Withdraw from streams where they lose balance

How are they judged?



Better microprediction

It is easy to incorporate any time series model

A model is just something that spits out 225 guesses of a future value

```
class BenchmarkCrawlerExample(SimpleCrawler):

    def __init__(self,**kwargs):
        super().__init__(**kwargs)

    def sample(self, lagged_values, lagged_times, **ignored):
        model.fit( lagged_values )
        half_width = 0.5/self.num_predictions
        prctls = np.linspace(half_width, 1-half_width, self.num_predictions)
        return [ model.invcdf(p) for p in prctls ]
```

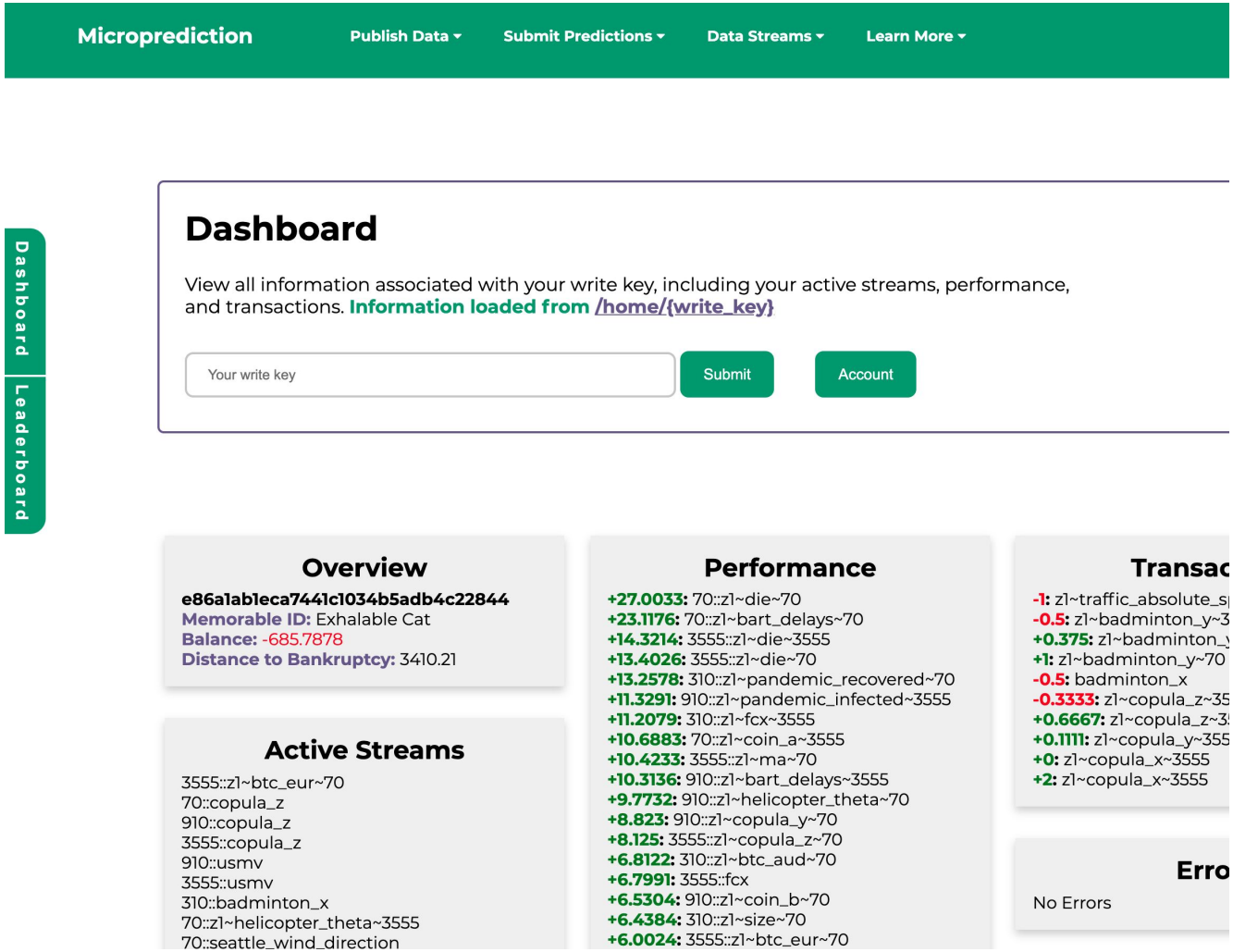
There are other abstractions (classes) you can use

Example: A t-digest used as a DistributionMachine inside SequentialStreamCrawler

```
from microprediction.config_private import STATESBOY_CAT
from tdigest import TDigest
from microprediction import SequentialStreamCrawler, DistributionMachine


class DigestMachine(DistributionMachine):

    def __init__(self):
        super().__init__()
        self.digest = TDigest()

    def update(self, value, dt=None, **ignored):
        self.digest.update(value)

    def inv_cdf(self, p):
        return self.digest.percentile(100. * p)


if __name__ == "__main__":
    crawler = SequentialStreamCrawler(write_key=STATESBOY_CAT, min_lags=500, machine_type=DigestMachine)
    crawler.set_repository(
        url='https://github.com/microprediction/microprediction/blob/master/crawler_examples/statesboy_cat.py')
    crawler.min_lags = 500
    crawler.run()
```

The Travelling Algorithm Problem

The Travelling Algorithm Problem (cont)



The Travelling Algorithm Problem (dumb way)

A crawler:

- Maintains a TODO list of what to predict
- Estimates when to predict
- Pulls a list of potential new streams

Now that part that is currently
not super clever:

- Picks at random
- Gives up at stop-loss

Performance

+35.7987: 70::z1~die~70
+28.3926: 3555::z1~copula_z~70
+24.7674: 70::z1~size~3555
+24.1433: 3555::z1~btc_eur~70
+22.064: 70::z1~bart_delays~70
+16.8854: 3555::z1~die~3555
+15.1565: 70::z1~coin_a~3555
+15.1038: 3555::fcx
+13.6464: 910::z1~copula_y~70
+13.0955: 3555::z1~copula_z~3555
+10.9293: 310::z1~bart_delays~70
+10.4233: 3555::z1~ma~70
+9.7736: 310::z1~traffic_speed_deltas~70
+9.6291: 910::z1~bart_delays~3555
+9.0671: 910::z1~coin_b~70
+8.9786: 3555::z1~die~70
+8.8225: 70::z1~coin_b~70
+8.0184: 3555::z1~copula_y~3555
+7.3876: 310::z1~fcx~3555
+5.3377: 910::z1~fcx~70
+3.8892: 910::z1~helicopter_psi~70
+3.8807: 310::z1~dvn~70
+3.2549:
910::z1~South_Australia_Electricity_Price~3555
+3.056: 310::z1~traffic_absolute_speed~70
+2.9035: 910::z1~cat~3555
+2.625: 310::z1~goog~70
+2.1289: 70::z1~seattle_wind_speed~70
+2.1007: 910::z1~c5_bitcoin~3555
+2.0632: 3555::z1~dvn~3555
+2: 910::z1~South_Australia_Electricity_Price~70
+0.4173: 910::z1~seattle_wind_direction~3555
+0: 310::z1~copula_x~70
+0: 70::z1~copula_z~70
+0: 70::z1~helicopter_theta~3555
+0: 70::coin_a
+0: 910::z1~copula_x~3555
-0.4545: 3555::z1~traffic_speed_deltas~3555
-1.0507: 310::z1~seattle_wind_direction~70
-1.051: 910::z1~copula_z~3555
-1.3249: 3555::c5_bitcoin
-1.5616: 3555::usmv
-2.1537: 70::seattle_wind_direction
-2.7451: 910::z1~c5_ethereum~70
-3.081: 310::z1~sq~70
-3.4996: 310::seattle_wind_direction
-3.629: 70::z1~helicopter_psi~3555
-4.455: 910::z1~c5_ripple~3555
-4.8213: 910::z1~helicopter_theta~70
-5.2059: 3555::cat
-5.6715: 910::z1~c5_cardano~3555
-6.136: 910::z1~qual~70

Game theory says:

Winner take all contests are the most efficient

Game theory says:

A standard way to model all-pay auctions is to assume that participants place a subjective value on winning, then decide how much to bid. We label the players in decreasing order of the values they place on the item, namely $v_1, v_2, \ldots, v_n$.

Translating an auction into a contest, we'd say that the participant that puts in the most effort is declared the winner. Participants incur a cost per unit effort which is common to all players. They seek to maximize

$$utility = expected \overbrace{subjective\ reward}^{v_i} - effort$$

Game theory says:

Equivalent formulation

Equivalently, we can assume that each player values the prize the same (and that value can be set to unity), but that players have different abilities (which multiply the effort they put in).

$$utility = expected \overbrace{reward}^{v=1} - effort \times \overbrace{production\ cost}^{inverse\ skill}$$

Game theory says:

It can be shown that the player who values the prize the most will choose effort randomly and uniformly between zero and the value v_2 that the second player ascribes to the prize.[7]

Conversely, the second player needs to keep the first player as honest as possible, and so will choose a random effort between zero and v_2 (to invest more would incur an obvious winner's curse).

It can be shown that the second player will make no effort at all with probability $1 - \frac{v_2}{v_1}$. Otherwise, the second player will draw uniformly in $(0, v_2)$.

Game theory says:

Between 1 and 2 participants per contest

Kaggle says:

Between 1,000,000 and 2,000,000 registered users per contest

I say:

Both nuts

The Travelling Algorithm Problem (better way?)

We view winning as the result of ability, work but also performance noise.

Marginal effort calculations

Green (wins 1/2 the time), yellow (wins 1/3) and blue (1/6) score distributions. Distribution of best in red.

Question: how hard should yellow try?

The Travelling Algorithm Problem

Marginal effort calculations (cont)

Evidence available to yellow:

1. Cost of "performance"
2. Frequency of winning.
3. Competitors entering and leaving.

Costs are associated with moving the performance location parameter. Examples:

1. Fixed cost of gathering and storing independent samples implies linear cost of precision.
2. Cost of hyper-parameter search.
3. Cost of buying additional precision from a "child" supplying ingredient data feed (middle man role).

The Traveling Algorithm Sub-Problem

The Discrete Horse Race Problem

Let $X_1, ..., X_n$ be discrete univariate contestant score-sassumed to take values on a lattice of equally spaced points. Let $X^{(k)}$ denote the k'th order statistic and in particular let $X^{(1)}$ denote the winning minimum score.

We define the implied state price for each contestant i as follows.

$$p_i = E\left[\frac{\iota_{X_i=X^{(1)}}}{\sum_k \iota_{X_k=X^{(1)}}}\right] \tag{3}$$

where ι is the indicator function. The price p_i is the expected payout in a game where we get

$$payoff = \begin{cases} 1 & if\ horse\ i\ wins \\ \frac{1}{2} & if\ tied\ with\ one\ other \\ \frac{1}{3} & if\ tied\ with\ two\ others \\ \dots \end{cases}$$

It reduces to the probability of winning if there are no ties.

The Traveling Algorithm Sub-Problem (key ideas)

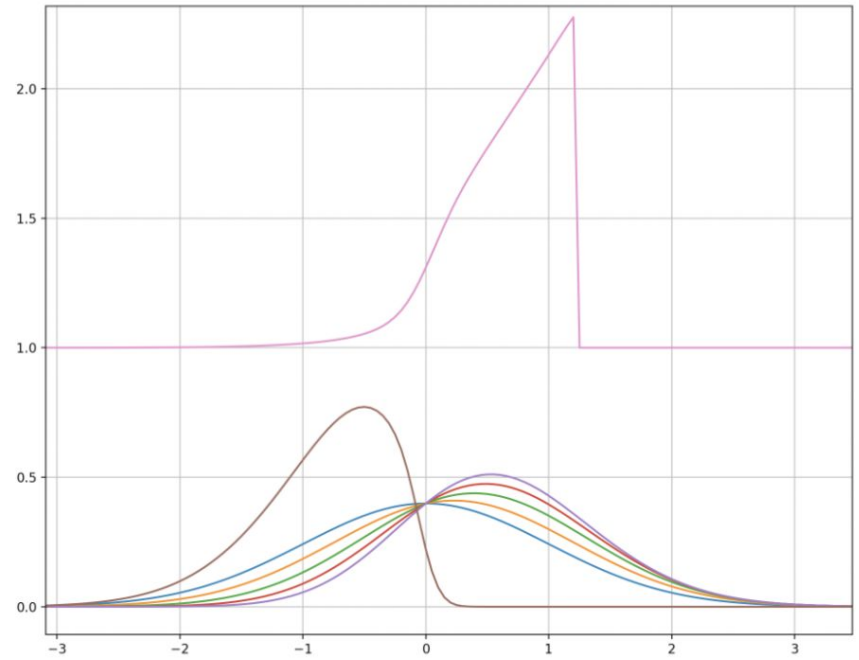
Compare one horse to all the rest

Characterize “all the rest” by first order statistic and mean conditional tie “multiplicity”

Develop a calculus that allows first order statistic and multiplicity to be quickly recalculated when one horse is removed.

Iterate horse relative ability

The Traveling Algorithm Sub-Problem (key ideas)



Example of tie multiplicity (conditional expected number of horses tying for first place, conditioned on a fixed performance) for a larger race with 25 contestants. An evenly spaced lattice with 500 points supports the scoring distributions.

The Traveling Algorithm Sub-Problem (key ideas - cont)

Multiplicity inversion

The multiplicity calculus can be inverted to determine the multiplicity for all-but-one of the horses.

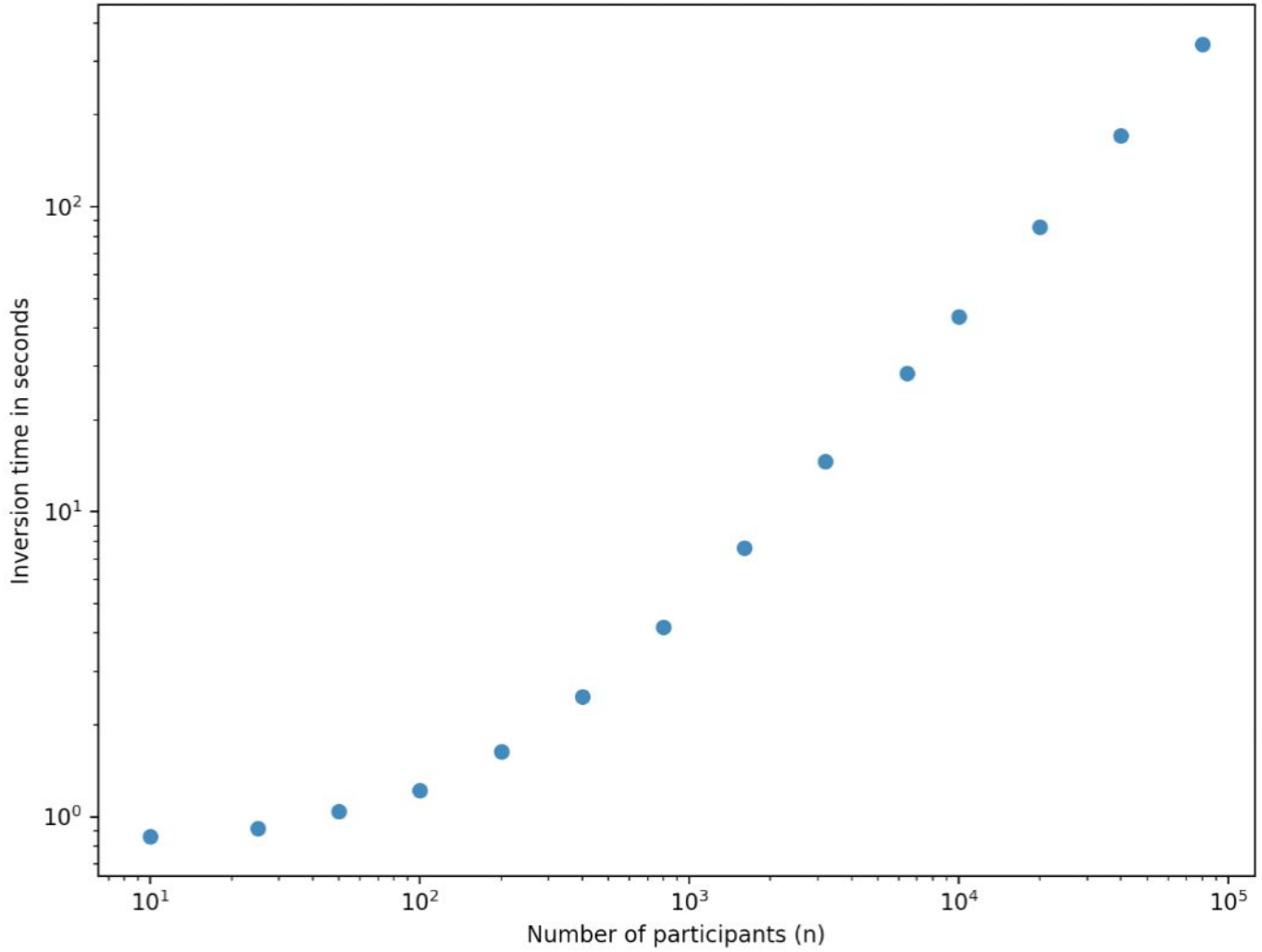
$$
\begin{aligned}
m_{\hat{i}}(j) &= \frac{numer}{denom} \qquad (9)\\
numer &= m(j)f_1(j)S_{\hat{i}}(j) + m(j)f_1(j)f_{\hat{i}}(j)\\
&\quad + m(j)f_{\hat{i}}(j)S_1(j) - m_1(j)f_1(j)S_{\hat{i}}(j)\\
&\quad - m_1(j)f_1(j)f_{\hat{i}}(j)\\
denom &= f_{\hat{i}}(f_1 + S_1)
\end{aligned}
$$

The Discrete Horse Race Solution - Numerical Results

Summary:

1. Convergence takes only a few iterations
2. A $100,000$ dimensional calibration problem can be solved in reasonable time

The Traveling Algorithm Problem Sub-Solution (cont)



The Traveling Algorithm Problem - Solution Summary

Marginal effort calculations (cont)

Approach:

1. Calibrate effort scale.
2. Calibrate relative location parameters of all players to contest winning frequency.
3. Compute marginal prize-money versus shadow price of performance improvement.
4. Quit sometimes.

Only (2) presents any genuine numerical challenge in the presence of a large number of competing algorithms and/or a large number of contests in which one might choose to participate.

Have fun !

```
from microprediction import MicroCrawler

if __name__ == '__main__':
    crawler = MicroCrawler(difficulty=9)
    crawler.run()
```

Join us for Friday virtual happy hours

Eric Lou · 1st

CS + Math Student at Stanford University

- Wrote the front end
- Winning crawlers

Rusty Conover · 1st

Experienced and innovative software executive with a track record of building businesses, platforms and applications.

- MUIDs in Java, Julia, Rust
- ZK-MUID proofs