Understanding exchange network dynamics with Python

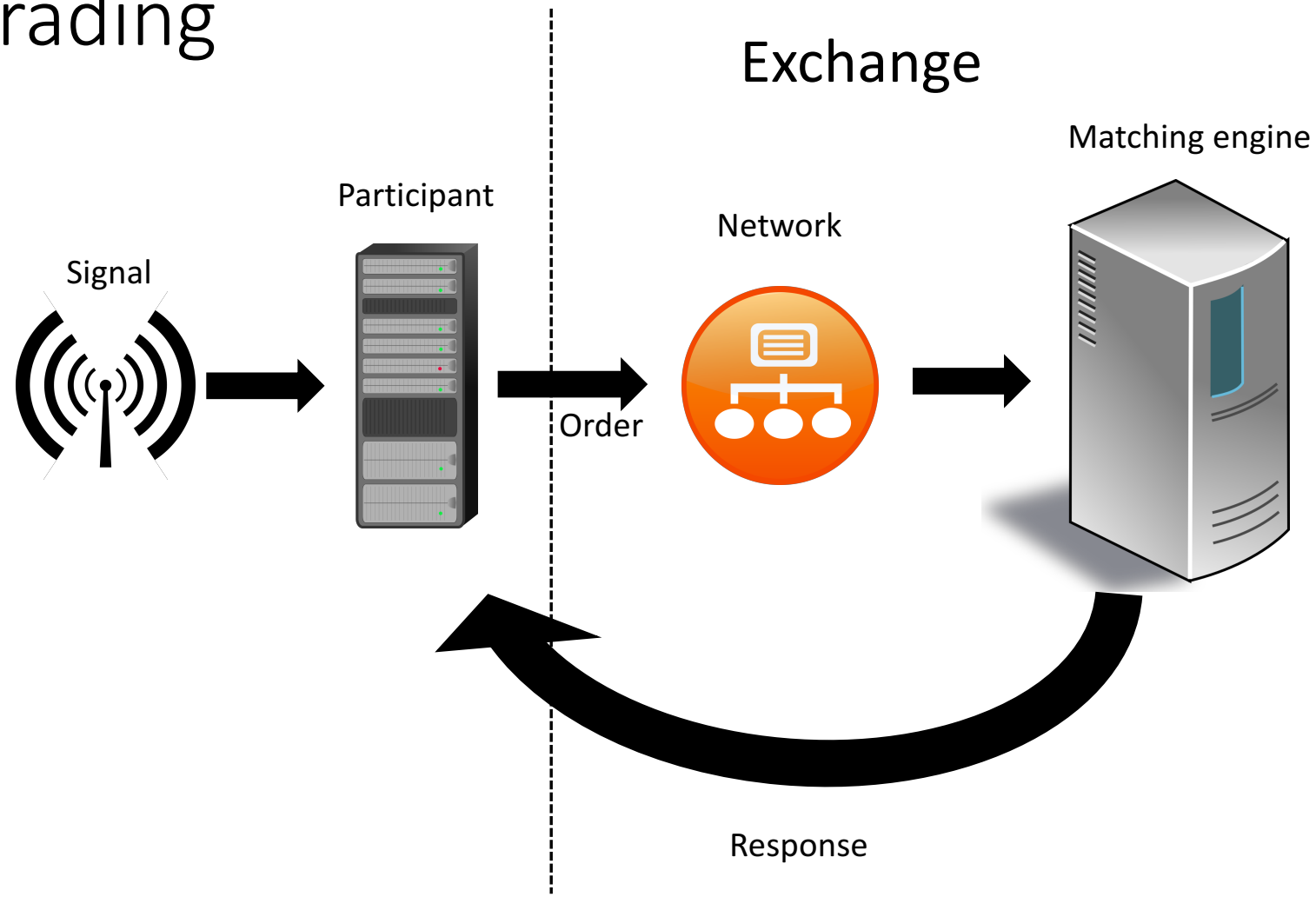
Omer Yuksel

Introduction

- Performance Analyst at IMC
- Technology-driven trading
- Amsterdam, Chicago, Sydney

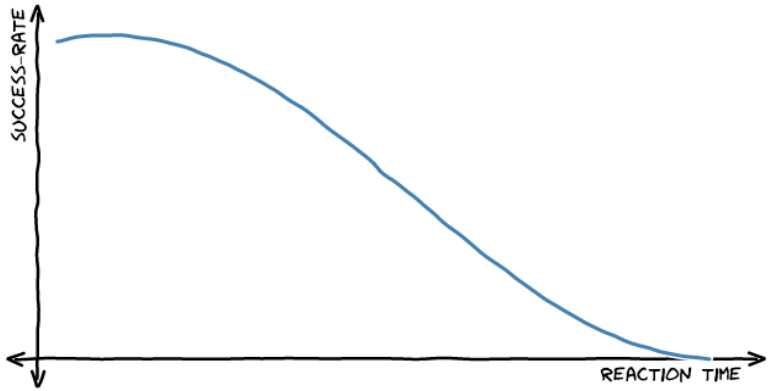
www.imc.com

Trading


Exchange
Matching engine
Participant
Network
Signal
Order
Response

Questions

- How often are we queuing?
- How much does queuing matter?
- How much should we improve our reaction time?



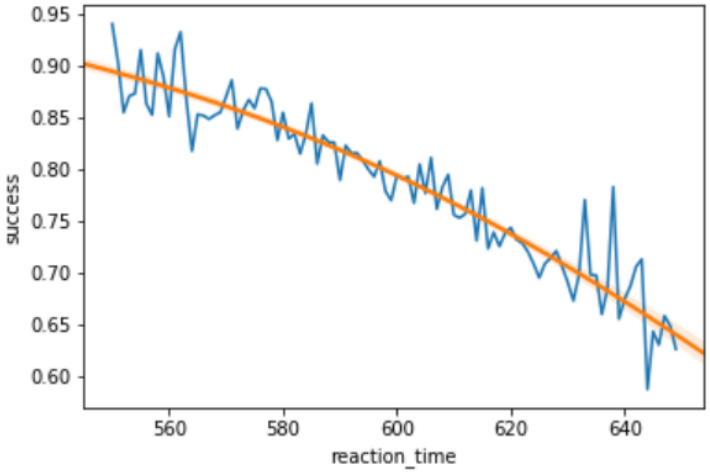
Methodology

Combine data with knowledge

- Data
  - Success/Failure
  - Reaction time
  - Time to reach matching engine

- Knowledge
  - Network structure
  - Components' behavior
  - Queuing effects

	ts	reaction_time	time_to_me	success	total_time_to_me
0	2019-06-03 09:00:00.042956193+02:00	606	5085	False	5691
1	2019-06-03 09:00:00.207588123+02:00	647	4500	False	5147
2	2019-06-03 09:00:00.276477343+02:00	633	4469	True	5102
3	2019-06-03 09:00:00.306361141+02:00	577	4459	True	5036
4	2019-06-03 09:00:00.389016054+02:00	599	4349	True	4948
5	2019-06-03 09:00:00.426910915+02:00	578	4515	True	5093
6	2019-06-03 09:00:00.552713100+02:00	572	4472	True	5044
7	2019-06-03 09:00:00.783978162+02:00	566	4680	True	5246
8	2019-06-03 09:00:00.870361884+02:00	606	4457	True	5063
9	2019-06-03 09:00:00.948479263+02:00	563	4394	True	4957



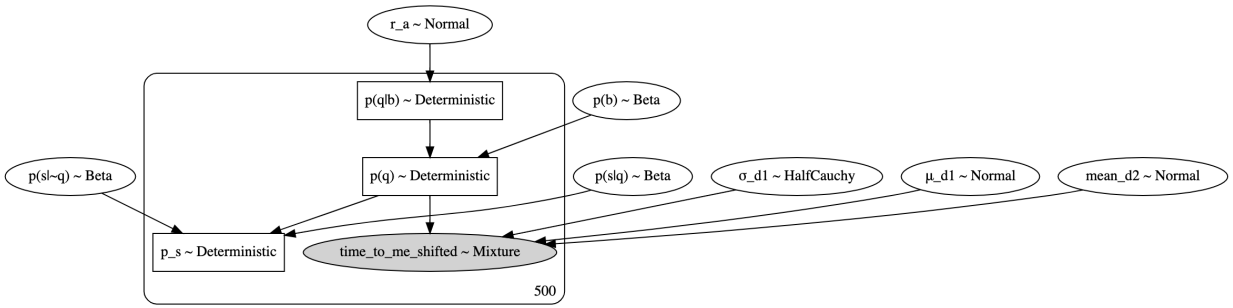
Approaches

- Probabilistic programming with PyMC3
  - Distributions and parameters
- Discrete-event simulations with SimPy
  - Code

PyMC3

- Bayesian probabilistic programming module

- Variables
  - Random
    - Free
    - Observed
  - Deterministic



High-level model

- Hyperparameters
  - Performance: $perf \sim Beta$
  - Probability of bursts: $p_b \sim Beta$
  - Probability of success despite queuing: $p_{s_a} \sim Beta$
- Model parameters
  - Prob. queuing with/without bursts
  - Prob. success
  - Time to matching engine with/without queuing
- Observed
  - Time to matching engine
  - Reaction time
  - Success failure
- Method: ADVI

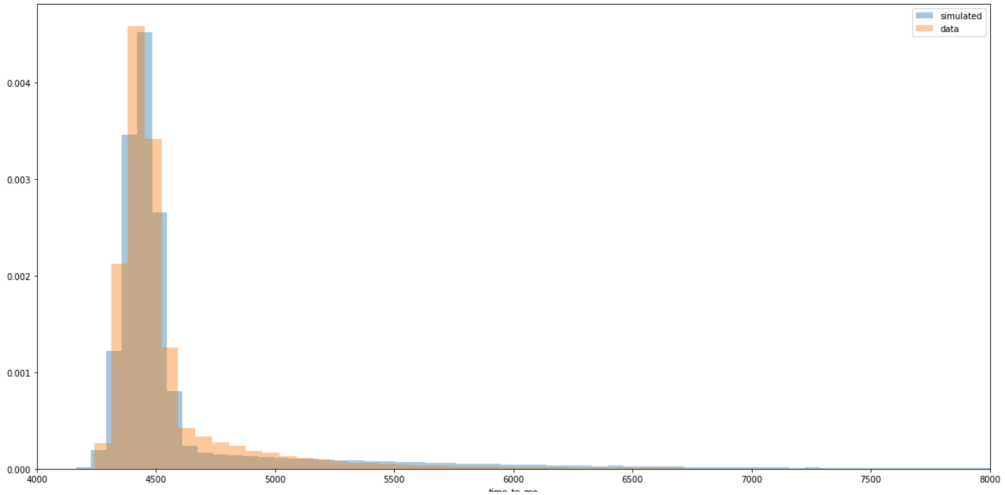
Success: $s \sim Binomial(p_s)$

Reaction time: $r \sim Normal(\mu_r, \sigma_r)$

Time to matching engine: $d_{me} \sim Mixture(d_1, d_2, w)$

Posterior sample

Time to matching engine:



Success rate:

Simulation = 78.8%
Data = 78.9%

Posterior sample

Reaction time

Queuing

Bursts

Success rate

Base success rate

	mean	sd	mc_error	hpd_2.5	hpd_97.5
r_mu	599.846909	1.566378	0.016985	596.739331	602.869329
r_sigma	22.180482	0.772145	0.008591	20.637166	23.673577
p_q	0.191285	0.029576	0.000252	0.135996	0.251843
p_b	0.255693	0.028668	0.000265	0.201732	0.313731
p_s	0.788471	0.024746	0.000210	0.738779	0.835984
p_s_a	0.424210	0.039147	0.000366	0.349996	0.502416

PyMC3 – What's next?

- Different models
  - Additive model for latencies
  - Explicit model for reaction window

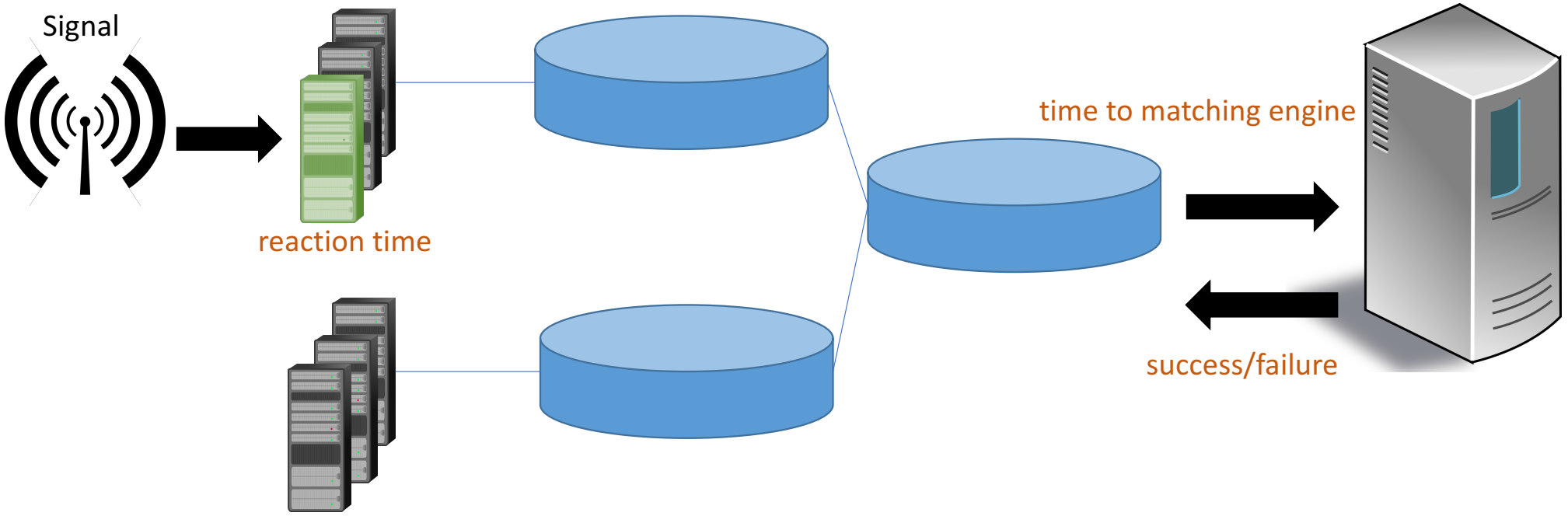
- MCMC vs VI

SimPy – Discrete Event Simulation

- Environment
- Event – “signal received"
- Process – participant, switch,
- Resources – queue, semaphore, …

https://simpy.readthedocs.io/en/latest/contents.html

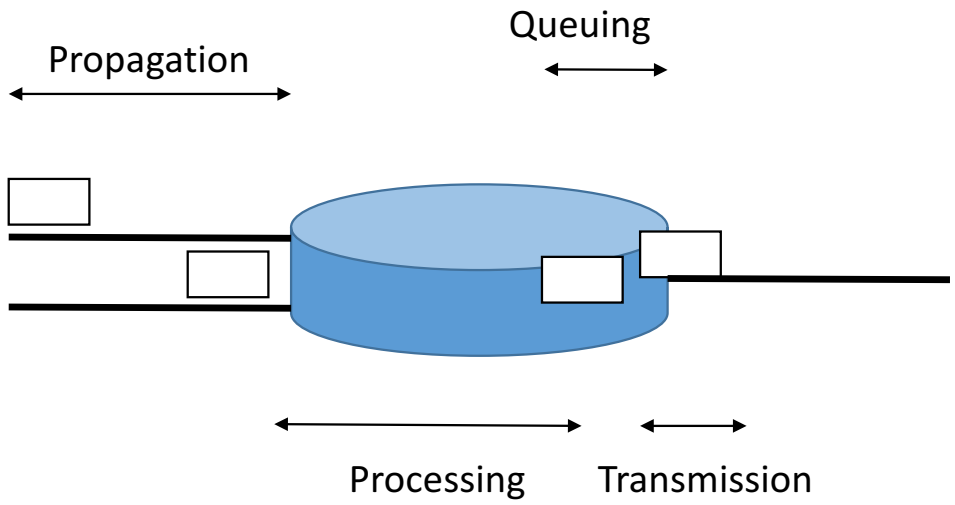
Network


Signal
reaction time
time to matching engine
success/failure

Delays

Speed of light, cable length

Number of bytes in the output buffer



Switch model and configuration

Line rate, packet size

Bursts and Queuing

- Bursts - multiple packets

- Queuing:
  - Transmission delays of the packets in front
  - Non-deterministic

Building blocks - Simulation

```
def run_simulation(duration):
    env = simpy.Environment()
    ...

    # Nodes
    matching_engine = MatchingEngine('matching_engine', env)
    switch = FIFOSwitch('Switch1', env, matching_engine, SWITCH_TO_MATCHING_ENGINE)
    alice = SimpleParticipant('Alice', env, switch, TIME_TO_SWITCH)
    bob = SimpleParticipant('Bob', env, switch, TIME_TO_SWITCH)

    env.run(duration)
```

Building blocks - Switch

```
class FIFOSwitch(NetworkNode):
    def __init__(self, name, env, destination, time_to_destination):…

    def run(self):
        while True:
            packet = yield self.incoming_packets.get()
            self.debug(f'{packet} arrived')
            self.env.process(self.process_packet(packet))

    def process_packet(self, packet):
        yield self.env.timeout(SWITCH_PROCESSING_DELAY)
        self.outgoing_packets.put(packet)

    def transmit_outgoing_packets(self):
        while True:
            packet = yield self.outgoing_packets.get()
            self.debug(…)
            self.send_to_destination(packet)
            yield (self.env.timeout(TRANSMISSION_DELAY))
```

Simulation output

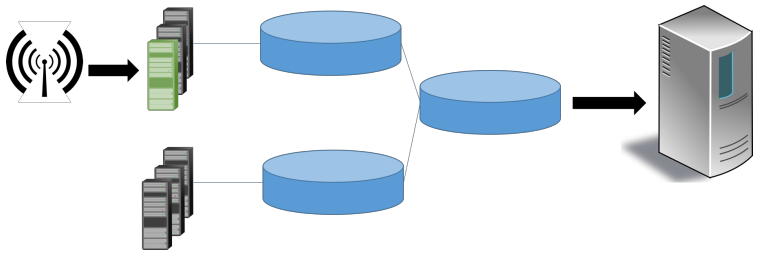
```
DEBUG [100000000] Alice - Signal received.
DEBUG [100000000] Bob - Signal received.
DEBUG [100000500] Alice - Sent packet to switch (Switch1)
DEBUG [100000500] Bob - Sent packet to switch (Switch1)
DEBUG [100001500] Switch1 - Packet (Alice) arrived
DEBUG [100001500] Switch1 - Packet (Bob) arrived
DEBUG [100002470] Switch1 - Transmitting packet. Remaining #packets in queue: 1
DEBUG [100002650] Switch1 - Transmitting packet. Remaining #packets in queue: 0
DEBUG [100003970] matching_engine - Received Packet (Alice).
DEBUG [100004150] matching_engine - Received Packet (Bob).
```

Expanding the model

- Non-deterministic switch
- Signal generator
- Bursts

Exchange model

```
env = simpy.Environment()

# Nodes
me = MatchingEngine('matching_engine', env)

switch_2 = AlmostFIFOSwitch('switch_2', env, me, 1000, switch_delay)

switch_11 = AlmostFIFOSwitch('switch_11', env, switch_2, 500, switch_delay)
switch_12 = AlmostFIFOSwitch('switch_12', env, switch_2, 500, switch_delay)

signal = BaseSignalGenerator('base_signal', env, signal_dist)

participant = Participant('participant', env, switch_11, 1000,
                          signal, reaction_time_dist_imc, window_dist)

burst = Burst('burst_1', env, switch_11, 1000, signal, reaction_time_dist_burst, num_reactions_dist)
burst_2 = Burst('burst_2', env, switch_12, 1000, signal, reaction_time_dist_burst, num_reactions_dist)
```



Exchange model

```
def run_simulation(duration, report_interval,
                           reaction_time_mean,
                           num_reactions_mean,
                           burst_reaction_offset,
                           window_size_loc):
    ...

    # Known distributions
    signal_dist = st.expon(scale=10 ** 8)
    switch_delay = st.truncnorm(loc=970 - REORDER_WINDOW, scale=50, a=-2, b=2)

    # Parameterized distributions
    reaction_time_dist_imc = st.truncnorm(loc=reaction_time_mean, scale=25, a=-2, b=2)
    num_reactions_dist = st.poisson(mu=num_reactions_mean)
    reaction_time_dist_burst = st.truncnorm(loc=reaction_time_mean + burst_reaction_offset,
                                            scale=70, a=-4, b=4)

    window_dist = st.lognorm(loc=window_size_loc, scale=10, s=7)

    ...
```

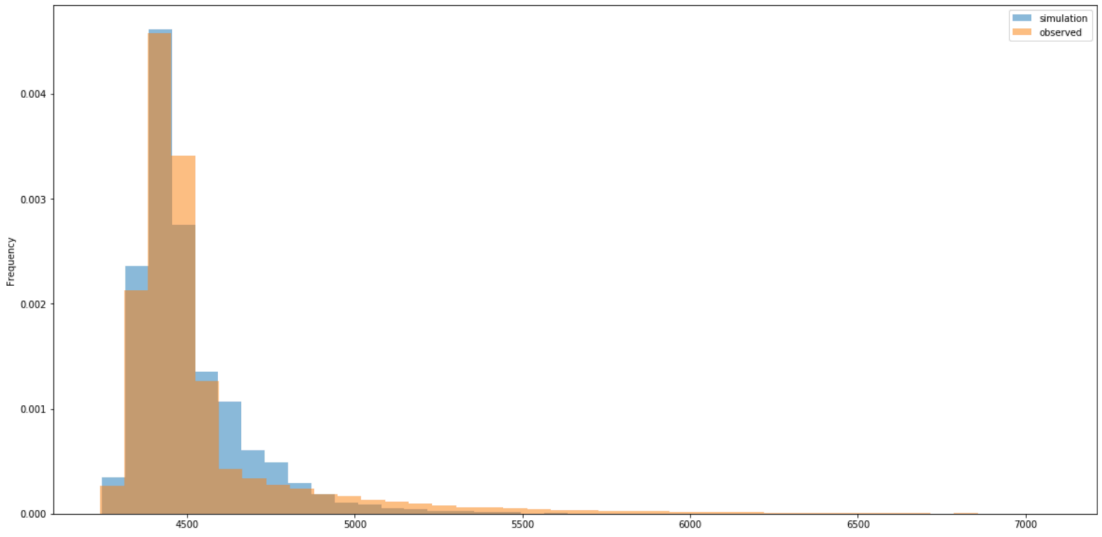
Fit model

- Parameters:
  - Burst behavior
    - #packets on average
    - Time offset
  - Reaction window
  - Average reaction time

- Cost function: Energy distance

Results

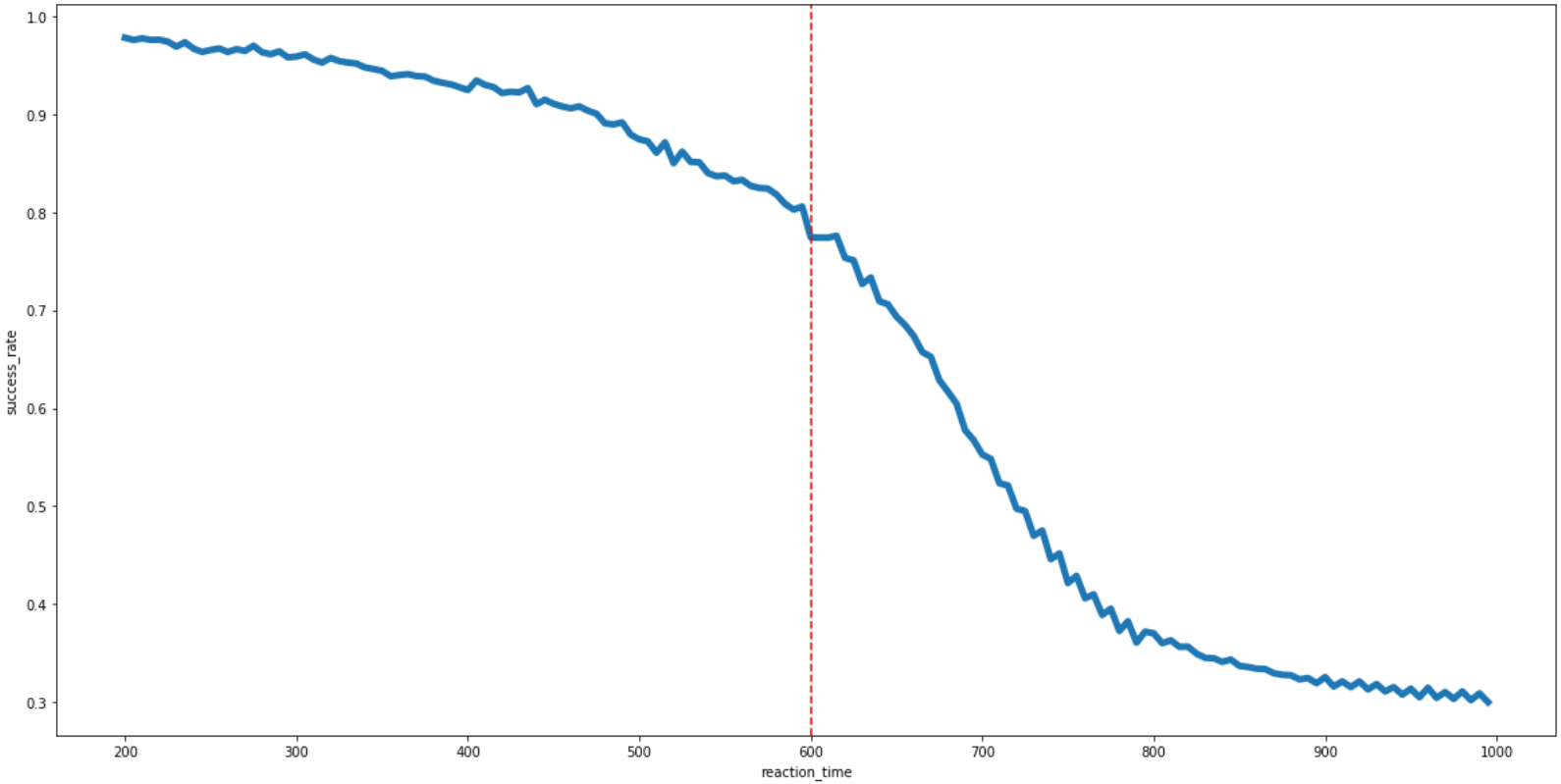
Time-to-matching engine:



Success rate:

- **Simulation: 78.4%**
- **Observed data: 78.9%**

Average reaction time vs success



When to use which?

- PyMC3
  - Concise
  - Many parameters
  - Fast with variational inference

- SimPy
  - Details
  - Discrete variables
  - Many moving parts
  - Scaling

Summary

- Incorporating knowledge
- Approaches
  - Discrete-event simulations
  - Probabilistic models

Questions

https://github.com/omersyuksel/pydata-london-2019