Evolving *ab initio* trading strategies in heterogeneous environments

David Rushing Dewhurst[*]
University of Vermont
Burlington, Vermont
david.dewhurst@uvm.edu

Yi Li
MassMutual Data Science
Boston, Massachusetts
yli63@massmutual.com

Alexander Bogdan
MassMutual Data Science
Boston, Massachusetts
abogdan@massmutual.com

Jasmine Geng
MassMutual Data Science
Boston, Massachusetts
jgeng@massmutual.com

ABSTRACT

Securities markets are quintessential complex adaptive systems in which heterogeneous agents compete in an attempt to maximize returns. Species of trading agents are also subject to evolutionary pressure as entire classes of strategies become obsolete and new classes emerge. Using an agent-based model of interacting heterogeneous agents as a flexible environment that can endogenously model many diverse market conditions, we subject deep neural networks to evolutionary pressure to create dominant trading agents. After analyzing the performance of these agents and noting the emergence of anomalous superdiffusion through the evolutionary process, we construct a method to turn high-fitness agents into trading algorithms. We backtest these trading algorithms on real high-frequency foreign exchange data, demonstrating that elite trading algorithms are consistently profitable in a variety of market conditions—even though these algorithms had never before been exposed to real financial data. These results provide evidence to suggest that developing *ab initio* trading strategies by repeated simulation and evolution in a mechanistic market model may be a practical alternative to explicitly training models with past observed market data.

CCS CONCEPTS

• **Computing methodologies** → **Artificial life**; *Agent / discrete models*; • **Applied computing** → *Economics*;

KEYWORDS

Agent-based models, financial markets, neuroevolution

ACM Reference Format:
David Rushing Dewhurst, Yi Li, Alexander Bogdan, and Jasmine Geng. 2020. Evolving *ab initio* trading strategies in heterogeneous environments. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3389806

[*]To whom correspondence should be addressed.

1 INTRODUCTION

Ab initio artificial intelligence—algorithms that are capable of learning or evolving master-level performance from a zero-knowledge baseline on a task normally performed by humans—is a long-held goal of the field in general [49]. Recent years have seen substantial progress toward this goal [49, 56, 57]. One particular area of interest is the development of algorithms that are able to trade financial assets without human supervision. The difficulty of this problem is enhanced by its fundamentally stochastic nature unlike the deterministic non-cooperative games of Go, chess, shogi, and Atari. of obvious economic incentives for intense competition amongst candidate solution algorithms: if an algorithm has a non-transient ability to make a statistically significant positive profit, the owner of that algorithm stands to reap large financial gains.

Though there has been prior work on *ab initio* trading strategies, such work has focused on small, homogeneous collections of agents that interact over shorter timescales than those considered in this study [18]. Trading strategies that use statistical and algorithmic methods more broadly are exceptionally common in the quantitative finance literature [31, 33, 34], and are used in practice with mixed results [25, 41, 42]. Evolutionary approaches to the development of trading strategies have focused on derivation of technical trading rules using observed market data as the training dataset and then backtesting the evolved rules on out-of-sample test data. [5, 27]. Likewise, evolutionary computation and agent-based models have been used extensively to study the macro properties of artificial asset markets rather than specifically studying the micro properties of individual trading strategies [17, 19, 22, 43–45, 51]. However, to our knowledge there has been no academic study of the possibility of developing *in vivo* trading strategies using purely *ab initio* methods—trading strategies that train or evolve using artificial data only, and then, at test time, trade using real asset prices—which is the approach that we take here.

We pursue this objective for two reasons: first, achieving this goal would be a useful step in the development of evolutionary "self-play" techniques in the context of stochastic games with many players [1, 36, 53]; and second, this would demonstrate that the development of profitable trading strategies could be realized by

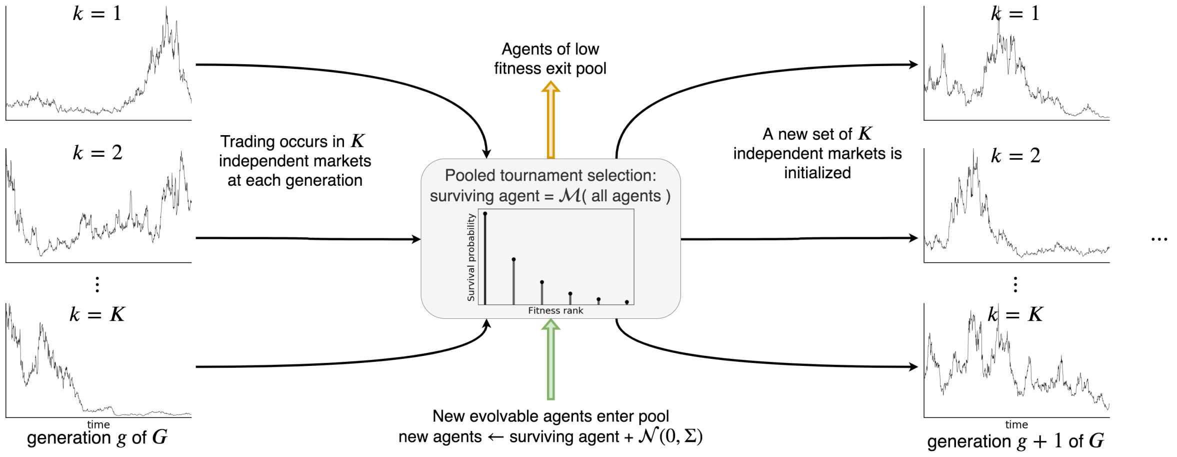
$k = 1$
$k = 2$
$k = K$
time
generation g of G
Trading occurs in K independent markets at each generation
Agents of low fitness exit pool
Pooled tournament selection: surviving agent = $\mathcal{M}$(all agents)
Survival probability
Fitness rank
New evolvable agents enter pool
new agents ← surviving agent + $\mathcal{N}(0, \Sigma)$
A new set of K independent markets is initialized
generation $g + 1$ of G

Figure 1: At each generation g of the evolutionary process, we initialize K independent markets in which agents (Sec. 2.2) interact via the matching engine described in Sec. 2.1. At the end of T timesteps, agents subject to evolution are pooled, selection is applied, and then new agents are introduced using the mechanism described in Sec. 2.3. The process then begins again in generation $g + 1$ and continues for a total of G generations.

attempting to simulate with increasing accuracy the underlying mechanisms of financial markets instead of by predicting future real market prices.

The paper proceeds as follows: in Sec. 2, we describe the theory and details behind our agent-based financial market model, including the design of the price-discovery (auction) mechanism, heterogeneous agent population, evolutionary algorithm (summarized graphically in Fig. 1), and method to convert evolved individuals into trading strategies; in Sec. 3 we summarize descriptive and quantitative results of the evolutionary dynamics and the performance of evolved trading algorithms backtested on real data; and in Sec. 4 we discuss these results and provide suggestions for future work.

2 THEORY AND SIMULATION

Our simulation methodology is based on an agent-based market model (ABM) composed of a matching engine and heterogeneous agents, which we describe presently [1]. We then outline the evolutionary mechanism, how it interfaces with the ABM, and the methodology by which we generate functional trading strategies from evolved agents.

2.1 Details of price discovery mechanism

At each timestep t, agents can submit orders to the matching engine, which attempts both to find an equilibrium price for that timestep and to match orders with one another so that exchange of shares for cash can occur. Orders are described by a three-tuple, $o = (s, N^{(o)}, X^{(o)})$, consisting of the desired side $s \in \{\text{buy}, \text{sell}\}$, the number of shares that the agent would like to purchase or sell $N^{(o)}$, and the requested price at which the agent would like to transact $X^{(o)}$. The matching engine collects all orders submitted to it and matches bid orders with ask orders using a frequent batch auction (FBA) mechanism [12, 13]. This mechanism is an alternative to the continuous double auction (CDA) mechanism that is in use in most securities markets. Both CDAs and FBAs are double-sided auctions, meaning that they match multiple buyers with multiple sellers of an asset at the same time, unlike auctions that match a single seller with multiple buyers (e.g., English, Japanese, or Dutch auctions) [6, 46, 48]. However, CDAs and FBAs differ fundamentally as CDAs operate in continuous time and FBAs operate in discrete time. CDAs match orders as they are received; if the order cannot be immediately executed, it is placed into an order book where it waits to be matched with a future incoming order. In contrast, an FBA collects orders in discrete time trading intervals. At the end of each trading interval, orders are sorted according to price preference (bids are sorted from highest to lowest price, while asks are sorted from lowest to highest). Matching then occurs in price-time priority order, meaning that bids with a higher price and asks with a lower price are matched first. Ties in price are broken by age, where older orders—orders that are already resting in the orderbook from previous batches—are matched first [2]. Orders submitted at timestep t that do not execute at timestep t remain in the orderbook for consideration in future time periods. Orders that remain in the orderbook—and hence are not matched with new, arriving orders—past a certain amount of time are considered "stale" by the matching engine and are removed. In our implementation, we set this period of time to be one day (or 100 time increments). Agents are able to observe the price X_t at time t and the volume of resting bid (b) and ask (a) orders in the orderbook at price level x at time t, written $D_b(x, t)$ and $D_a(x, t)$. All statistics used by agents in calculating side of book, price level and number of shares to submit are functions of these observable random variables and of the agents' own internal state, which we describe in the next section. We display an example orderbook, along with the corresponding price trajectory, in panel A of Fig. 2.

[1] All source used in this project is available from the authors upon reasonable request.

[2] The price discovery algorithm used in our matching engine implementation is a modified version of the logic used in the Australian Securities Exchange's matching engine [21]. An open-source implementation of the matching engine is at https://gitlab.com/daviddewhurst/verdantcurve.

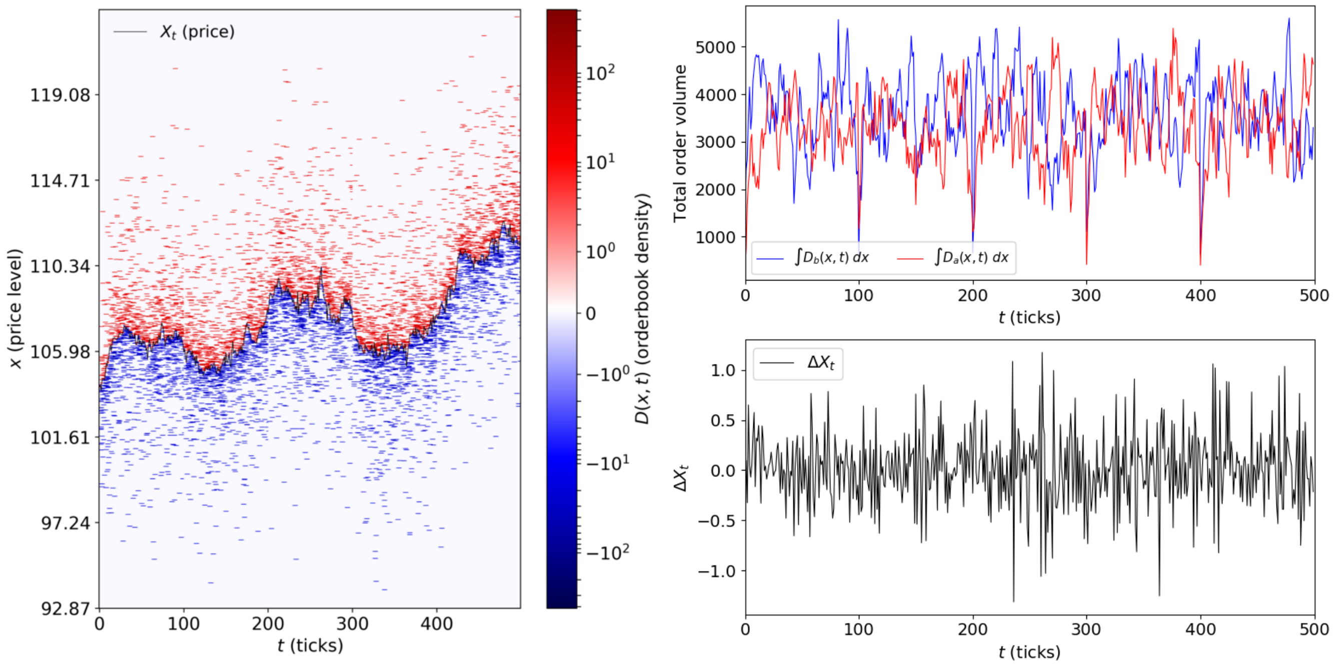

Figure 2: In panel A, we display an example orderbook corresponding with a very simple market simulation along with the resulting asset price time series. We denote bid interest (plotted in blue) by negative numbers (corresponding with positive $D_b(x,t)$ later) and ask interest (plotted in red) by positive numbers (corresponding with positive $D_a(x,t)$ later). We display the time series of total bid and ask interest in panel B and the time series of ΔX in panel C. Differences in total bid and ask interest are strongly associated with changes in level of ΔX. The periodicity of large drops in $\int D_b(x,t)\,dx$ and $\int D_a(x,t)\,dx$ in panel B is due to end-of-day orderbook clearing by the matching engine.

2.2 Heterogeneous agents

Our agent population is heterogeneous, comprised of seven qualitatively distinct varieties (species) of agents that analyze statistical behavior of asset prices differently and concomitantly exhibit divergent trading behavior. We do not apply evolutionary pressure to six out of seven of these species; we thus separate the descriptions of the agents below into those not subject to evolutionary pressure (static agents) and those that do evolve.

2.2.1 Static agents. There are six types of agents in our model to which we do not apply evolutionary pressure. We give a brief overview of them presently.

- *Zero-intelligence* (ZI): Inspired by work on statistical aspects of asset markets [30, 32], these agents submit a random bid or ask order with order price uniformly distributed around the last market price and number of shares Poisson-distributed around a fixed value (here set to be 100 shares).
- *Zero-intelligence priceless* (ZIP): identical to zero-intelligence agents, except these agents submit "market" orders — orders that do not have a price but rather have first-priority execution and execute against the highest bid (for a market ask) or lowest ask (for a market bid).
- *Momentum* (MO): Momentum trading agents postulate that prices that have recently risen will continue to rise (and that prices that have recently fallen will continue to fall) [7, 15]. Our agents submit bid orders if the change in price is above some static positive threshold and ask orders if the change in price is below a symmetric negative threshold.
- *Mean-reverting* (MR): these agents postulate a return to some mean value of the asset price [26, 60]. When the asset prices move above a rolling mean value, these agents submit ask orders; when the price moves below the rolling mean, they submit bid orders.
- *Market-making* (MM): market-making strategies attempt to profit off of small price imbalances on either side of the orderbook; in doing so, they provide liquidity to the asset market [24, 50].
- *Fundamental-value* (FV): these agents have a certain fixed price set at which they "believe" the asset is fairly valued; if the asset price rises above that fundamental value, they submit ask orders, while if the asset price falls below it, they submit bid orders. Note that this approach differs from that of MR traders since FV traders' valuations of the asset do not change over time.

We give more details on the implementation of these strategies in the supplementary information. Though this typology of strategies substantially overlaps with that introduced in the context of modern-day futures markets [40], it also exhibits some differences. In particular, we do not implement a pure "high-frequency trader" agent since this does not make sense in the context of an FBA

[2, 13]. We implement a wide variety of strategies so that, in order for evolving agents to achieve high fitness, they must perform well in many different environments. We believe this will increase the likelihood of high-fitness agents performing well when confronted with real price data on which to trade, since real asset markets are also composed of heterogeneous agents [40].

2.2.2 Evolvable agents. We model more expressive agents subject to evolutionary pressure as deep neural networks, denoted by f_θ (we will omit the vector of parameters θ when it is contextually unnecessary). These neural networks take as inputs changes in price, changes in total orderbook volume, and changes in internal state (cash, shares held, and profit) and output a three-tuple of side (bid or ask), number of shares in the order, and price level of the order:

$$(s_t, N_t^{(o)}, \Delta X_t^{(o)}) = f(\Delta X, \Delta\hat{V}^{(b)}, \Delta\hat{V}^{(a)}, \Delta c, \Delta N, \Delta\pi) \tag{1}$$

The change in asset price from $t-1$ to t is given by $\Delta X_t = X_t - X_{t-1}$, while the change in cash (Δc), shares (ΔN), and profit ($\Delta\pi$) are defined analogously. The total bid and ask interest in the orderbook at time t are given by $\hat{V}_t^{(b)} = \int_0^{X_t} D_b(x,t)\ dx$ and $\hat{V}_t^{(a)} = \int_{X_t}^{\infty} D_a(x,t)\ dx$ respectively; we then have $\Delta\hat{V}_t^{(b)} = \hat{V}_t^{(b)} - \hat{V}_{t-1}^{(b)}$ and $\Delta\hat{V}_t^{(a)} = \hat{V}_t^{(a)} - \hat{V}_{t-1}^{(a)}$. The neural network is a four-layer feed-forward model. The two hidden layers have 20 and 10 neurons respectively, for a total of 383 evolvable parameters in each neural network [3].

It is an analytical convenience to consider a single generation of the evolutionary process described in Sec. 2.3 as a draw from a stochastic function $\mathcal{G}(\alpha, \mathcal{M})$, where α describes the specification of the agents in the model and $\mathcal{M}$ is the evolutionary mechanism (selection and mutation); we will describe these parameters in some detail in Sec. 2.3. This function yields orderbooks and price time series; each call to the function yields a tuple of bid and ask order density and a price time series,

$$(D_b(x,t), D_a(x,t), X_t) = \mathcal{G}(\alpha, \mathcal{M}). \tag{2}$$

This way of looking at the process makes it easy to express Monte Carlo estimates of theoretical quantities and provides the theoretical basis for conversion of evolved agents into trading algorithms as we outline in Sec. 2.4. We can re-express the above integrals as Monte Carlo estimates (which is how we compute them in practice) so that

$$\hat{V}_t^{(b)} \approx \sum_{\text{observed bids } x'} D_b(x', t) \tag{3}$$

and

$$\hat{V}_t^{(a)} \approx \sum_{\text{observed asks } x'} D_a(x', t), \tag{4}$$

where $D_b(x,t)$, $D_a(x,t)$, and X_t are given by Eq. 2. As a visual reference point, in Panel B of Fig. 2 we display an example realization of Monte Carlo-approximated $\Delta\hat{V}^{(b)}$ and $\Delta\hat{V}^{(a)}$, while in panel C of this figure we display the corresponding ΔX.

2.3 Evolutionary dynamics

We provide a summary of the evolutionary dynamics in Fig. 1. We first describe the selection and mutation mechanism $\mathcal{M}$, since this mechanism changes on the objective of the simulation, and then describe the simulation in general. We set the evolutionary mechanism $\mathcal{M}$ to be either tournament selection-based or the identity (no evolution): we use the tournament selection mechanism when we are attempting to evolve agents of high fitness, while we use the identity mechanism when we are generating empirical market statistics for use with real data, as described in Sec. 2.4.

The tournament selection mechanism is standard [11], designed as follows: given a population of evolvable agents, at the end of each generation a tournament of τ agents is selected from the population at random. We set $\tau = 17$. These agents are sorted by fitness—their total profit π in the market simulation of the past generation—so that $\pi_{(1)} \geq \pi_{(2)} \geq \cdots \geq \pi_{(\tau)}$. Agent (i) is selected to remain with probability $p(1-p)^{i-1}$, where we set $p = \frac{1}{2}$; the remaining agents are discarded. A total of $\tau - 1$ new agents are initialized with the parameters from the selected agent (i), denoted by $\theta_{(i)} = (\theta_{(i),1}, ..., \theta_{(i),L})$ where the agent has a total of L parameters. These new parameters are then subjected to centered Gaussian mutation; the parameters of new agent i' are given by $\theta_{i'} = \theta_{(i)} + z_{i'}$, where $z_{i'} \sim \mathcal{N}(0, \gamma^2\Sigma)$ and we set $\gamma = 0.1$. The covariance matrix Σ of the Gaussian is diagonal, with $\Sigma_{\ell\ell} = \text{Var}(\theta_{(i),\ell})$. The $\tau - 1$ agents are added back into the entire population of evolvable agents for use in further generations, described in the next paragraph.

At each of $g = 1, ..., G$ generations, we initialize $k = 1, ..., K$ independent markets. We set $K = 24$ and $G = 100$. In each market, we initialize N_A agents with agent parameter vector distributed as $(\alpha_{a,k})_{a\in A} = \alpha_k \sim p(\alpha)$, where A is the set of agent types outlined in Sec. 2.2. Given a drawn α_k, there are $\alpha_{\text{ZI},k}$ zero intelligence agents, $\alpha_{\text{MO},k}$ momentum agents, and so on. The probability distribution $p(\alpha)$ is a joint distribution over agent types that factors as a uniform distribution over the number of NN agents and a multinomial distribution over the number of other agents. Given a number of neural network agents $\alpha_{\text{NN},k}$ drawn uniformly at random between NN_{min} and NN_{max}, the remaining $N_A - \alpha_{\text{NN},k}$ agents are drawn from a multinomial distribution with probabilities $\rho_a = \frac{1}{\text{\# of non NN agent types}} = \frac{1}{6}$. We set $\text{NN}_{\text{min}} = 2$ and $\text{NN}_{\text{max}} = 10$. Within each market, agents interact *vis-à-vis* the matching engine described in Sec. 2.1, trading for a total of T timesteps within each generation. We set $T = 500$ and set the number of trading timesteps per day equal to 100, as outlined in the previous section. After T timesteps, the population of neural networks is pooled—removed from each simulation and collated into one set—and the evolutionary mechanism $\mathcal{M}$ is applied to all $\sum_{k=1}^{K} \alpha_{\text{NN},k}$ neural networks, generating a (partially) new population. The new population of neural networks is randomly partitioned into K sets, each corresponding to a new independent market simulation. The k-th market has $\alpha_{\text{NN},k}$ neural networks and $N - \alpha_{\text{NN},k}$ static agents drawn from the multinomial distribution. The simulation and evolution process then begins again in generation $g + 1$.

[3] Number of parameters is equal to number of parameters in the weight matrices plus the number of parameters in the bias vectors = 350 + 33.

2.4 From evolved agent to trading algorithm

We convert evolved neural networks into executable trading strategies that we subsequently backtest on real financial data. The major impediment to simply using the evolved networks as trading strategies is the lack of readily-available orderbook information for real financial markets: though such information is available for sale, it is prohibitively expensive to purchase [59]. Instead, we use orderbook data generated by $\mathcal{G}(\alpha, \mathcal{M})$ as a surrogate for real orderbook data. Orderbook data is an important input into the algorithms because it has been shown to carry non-zero information about future prices and is useful in making profitable short-term trading decisions [14, 35, 55]. If $\mathcal{G}(\alpha, \mathcal{M})$ is an accurate simulacrum of a real financial market's orderbook-generating process, the values of $\Delta\hat{V}^{(b)}$ and $\Delta\hat{V}^{(a)}$ generated by the ABM, given an observed value of ΔX from a real market, should be statistically similar to changes in resting bid and ask volume that exist in the real market.

We must simulate $\Delta\hat{V}^{(b)}$ and $\Delta\hat{V}^{(a)}$ that correspond with the observed change in price ΔX. To do this, we first run many simulations of $\mathcal{G}(\alpha, \mathrm{id})$ (the generative model with no evolutionary mechanism). Then, given ΔX from the real asset market, we draw multiple $(\Delta\hat{V}^{(b)}, \Delta\hat{V}^{(a)})$ pairs from their empirical joint pdf conditioned on ΔX, which is generated by the simulations of $\mathcal{G}(\alpha, \mathrm{id})$:

$$(\Delta\hat{V}^{(b)}, \Delta\hat{V}^{(a)}) \sim \hat{p}_{\mathcal{G}(\alpha,\mathrm{id})}(\Delta V^{(b)}, \Delta V^{(a)} | \Delta X), \tag{5}$$

We then evaluate f using the conditional expectation of these values. This new "marginalized" algorithm is given by

$$\begin{aligned} &f_{\mathrm{marg}}(\Delta X_t, \Delta c_t, \Delta N_t, \Delta \pi_t) \\ &\quad = f(\Delta X, E[\Delta\hat{V}^{(b)}|\Delta X], E[\Delta\hat{V}^{(a)}|\Delta X], \Delta c, \Delta N, \Delta\pi), \end{aligned} \tag{6}$$

where the expectations are taken under the pdf displayed in Eq. 5. This marginalized algorithm returns a side, change in shares, and change in price:
$(s_t, N_t^{(o)}, \Delta X_t^{(o)}) = f_{\mathrm{marg}}(\Delta X_t, \Delta c_t, \Delta N_t, \Delta \pi_t)$. Since we are backtesting the trading algorithm and hence it is impossible to perform price discovery, we do not use $\Delta X_t^{(o)}$. We just simulate the execution of a market buy ($s_t = 1$) or sell ($s_t = -1$) order for $N_t^{(o)}$ spot contracts of the asset and then update the algorithm's internal state. As in the agent-based model, we allow the algorithm to sell short so that N_t may be negative.

We implement three basic risk management routines that supervise the execution of the algorithm. These routines act as "circuit breakers" to halt the algorithm's operation if certain risk limits are reached and consist of two versions of the traders' adage "cut your losses but let your winners run," ensuring that maximum loss is capped at some user-set limit, and one leverage limit routine that halts execution if the algorithm is long or short a certain large number of contracts (we set this number equal to 150) [3]. (The interested reader is referred to the supplementary information for more detail.) Though these routines pale in comparison with real risk-management software used in algorithmic trading [20, 29, 58], we believe that they are sufficient for the purposes of this work.

3 RESULTS

3.1 Evolutionary dynamics

We ran 100 independent simulations of the entire process outlined in Sec. 2.3. This resulted in a total of 240,000 (= 24 independent markets per generation × 100 generations × 100 independent simulations) conditionally independent market simulations from which to sample evolved neural networks for validation and testing on real financial asset data. At each generation of each simulation, we saved the best individual for possible further use.

Evolved neural networks (NN) quickly became the dominant species of trading agent, though they were not profitable at $g = 0$ (at which point they were simply random neural networks with normally-distributed weights and biases). Fig. 3 displays average (solid curves) and median (dashed curves) wealth trajectories for each agent type; we calculated these statistics over all simulations at that generation. The average and median wealth of NN agents increased quickly until about $g = 10$. It then increased more slowly until about $g = 40$, when it plateaued. Concomitant with the rise in average and median NN wealth was a decline in the wealth of nearly every other agent type. In particular, though fundamental value (FV) agents started out as the most profitable agent type (due to their strong beliefs about "true" asset values and market power; in early generations they had the ability to collectively set market price), they became the second-worst performing agent type as NN agents became more profitable. It is notable that momentum trading agents were the only static agent type that was still able to make positive profits during multiple sequential later generations (in particular $g > 30$), long after all other static agents had become unprofitable on average. This is consistent with the finding that real asset prices may exhibit momentum effects and that trading strategies based on exploiting this momentum may result in positive expected profit [38].

As evolution progressed through generations, statistical properties of asset price time series X_t changed significantly. The mean square deviation (MSD) of X_t in generation g, defined by the exponent γ_g in the relationship $E_g[(X_t^{(g)} - \mu_t)^2] \propto t^{\gamma_g}$ where μ_t is the intertemporal mean of $X_t^{(g)}$, began in the sub- or normally-diffusive region ($\gamma_g \leq 1$) but quickly rose to $\gamma_g \approx 1.8$ near $g = 10$ and remained there for the remainder of evolutionary time. We display the MSD of asset prices by generation in Fig. 4. MSD that grows superlinearly with time is termed anomalous superdiffusion [9, 52] and is commonly observed in real asset markets [23, 28, 47]. This finding provides evidence that observed asset price superdiffusion in real asset markets is partially driven by purely endogenous evolutionary dynamics.

3.2 Validation and testing of evolved strategies

We chose a subsample of the neural networks that we extracted from the market simulations for consideration as algorithmic trading strategies to be used on real data. Though all evolved neural networks that we saved had high fitness in the context of the agent-based model, we hypothesized that it would not be the case that all of them would have high fitness when backtested on observed asset price data. We selected the high-fitness neural networks saved at generation $g = 10$, the set of which we will denote by $\mathcal{A}_{10}$, for

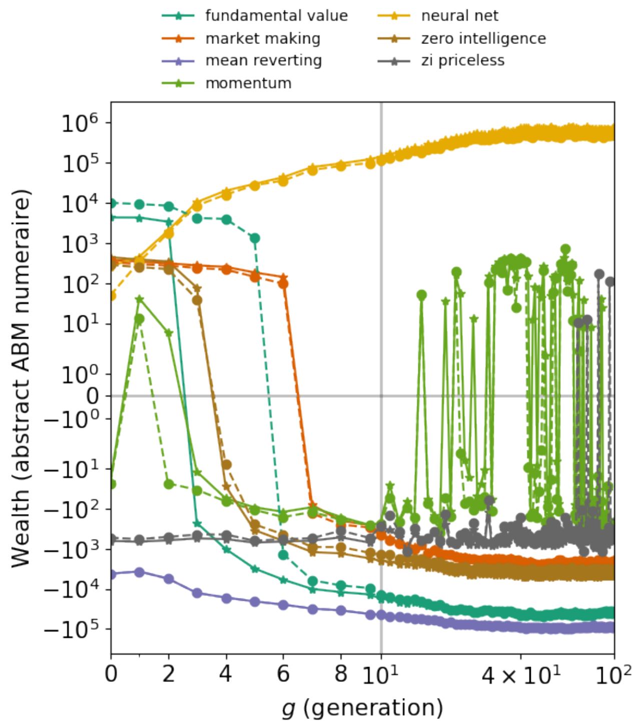

Figure 3: Evolving neural network agents quickly dominated all static agents. The average wealth of neural network agents increased until about $g = 40$, where it plateaued while the average wealth of other static trading agents remained largely flat or decreases over time. In particular, mean-reverting and fundamental-value traders suffered large average wealth losses, ' even though fundamental-value traders started as the most profitable agent type. We also find that a static momentum trading strategy was, on average, the strategy that was least dominated by evolving neural networks and could actually be sustainably profitable for multiple generations. This result corresponds with the finding that a momentum-based strategy can be profitable over nontrivial timescales in real financial markets [39].

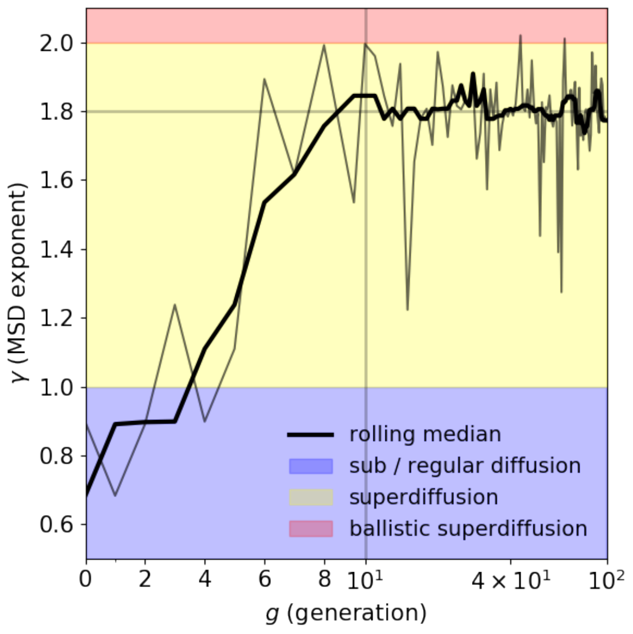

Figure 4: Asset price superdiffusion emerges as a byproduct of evolutionary pressure. Superdiffusion is defined by a superlinear relationship between mean squared deviation of a time series and time itself. At each generation g, we fit a model of the form $E[(X_t^{(g)} - \mu_t)^2] \propto t^{\gamma_g}$ and plot the resulting γ_g as a function of generation g. This exponent of dispersion stabilized at roughly $\gamma_g \simeq 1.8$ after approximately 10 generations of evolution (indicated by the vertical black line at $g = 10$), which influences our selection of $g = 10$ for validation and testing of evolved strategies on real data.

two reasons. First, this was the approximate "elbow" of $\log_{10} \pi$, as displayed in Fig. 3; at generations later than approximately $g = 10$, NN agents exhibited decreasing marginal $\log_{10} \pi$. Second, this generation was the point at which the exponent of the MSD of asset prices appeared to stabilize at $\gamma_g \approx 1.8$.

The asset prices produced by the interaction of agents in the ABM do not exhibit geometric (multiplicative noise) dynamics, but rather arithmetic (additive noise) dynamics. Though many real financial assets do exhibit geometric- or geometric-like dynamics [10], other assets, such as foreign exchange (FX) spot contracts, typically display quasi-arithmetic dynamics instead [37]. We thus test our evolved neural networks on FX spot rate data, eight and a half (January 2015 through July 2019) years of millisecond-sampled EUR/USD and GBP/USD spot exchange rate data sourced from an over-the-counter trading venue [4]. We do not implement transaction costs in our backtesting as, if these are constant, their marginal incidence on profit decreases as the amount of leverage (net number of contracts traded) increases. We separate this dataset into a validation (2010 - 2015) and testing (2015 - 2019) split. Although this split is unnecessary in the context of overfitting to data (the neural networks are static during validation and testing since their weights do not update in the absence of evolutionary pressure), separating into validation and test sets lowers the probability of false positive discovery of high-performance trading algorithms. We used the top k algorithms in $\mathcal{A}_{10}$, ranked by total profit accumulated by trading on validation data, to trade on the test dataset. If these algorithms accumulated high profit on the validation dataset by chance, it is likely that they would not be profitable on the test dataset. We set $k = 5$ and denote the set of "elite" trading strategies by $\mathcal{A}_{\text{elite}}$.

To create trading algorithms from the evolved neural networks, we followed the procedure described in Sec. 2.4. We resampled the spot FX rate time series at the 10s resolution, setting as X_t the mean price during that 10s interval multiplicatively rescaled by a constant $(100/X_0)$ so that the price on the first second of each month

[4]The data is available from https://www.truefx.com/, which sources it from the Integral OCX ECN.

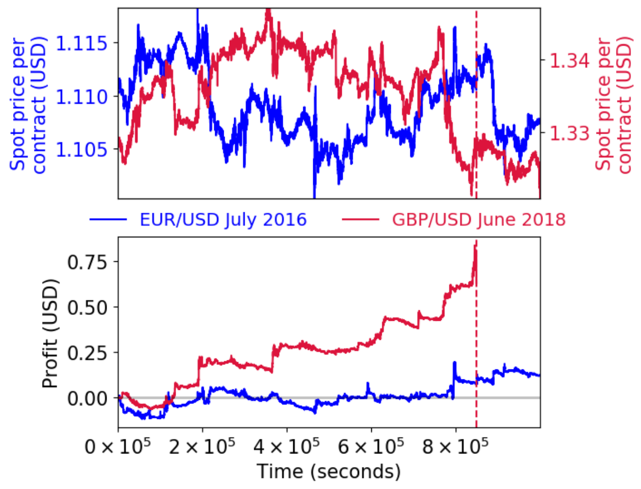

Figure 5: Elite evolved trading algorithms are able to obtain positive profit under a wide variety of backtested trading conditions. These price time series display both large increases and decreases during this time period, as well as regions of relatively low and high volatility. Despite these varied conditions, an elite evolved algorithm was able to capture positive profit (shown in the blue curve) over this time period, showing large gains in profit during both price drawdowns and ramp-ups. The vertical dashed line in each subplot indicates that the risk management system halted the execution of the trading algorithm. In this case, this occurred because the algorithm attempted to exceed the user-set leverage limit.

was equal to 100. (Profit and loss numbers were calculated using the true observed market price, not the rescaled price used as an input to the algorithm.) Agents traded during the first 10^6 seconds (approximately 16.2 days[5]) of each month, a number of timesteps that we chose arbitrarily to avoid possible edge effects occurring at the end of the month. We will refer to one 10^6s time interval of trading on a single spot rate (EUR/USD or GBP/USD) as a single trading episode. In Fig. 5, we display two example time series of spot FX rate (EUR/USD during July of 2016 and GBP/USD during June of 2018) and corresponding profit made by an evolved neural network during these trading episodes. In these examples, though the spot rate fluctuates considerably and has $|X_T - X_0| < 0.01$ USD/contract, the profit time series increased fairly steadily throughout each period of time.

Evolved strategies differed significantly from random neural network strategies. We compared the distribution of profit on the validation dataset by $\mathcal{A}_{10}$, profit on the test dataset by $\mathcal{A}_{\text{elite}}$, and profit on the test dataset by 20 random neural network agents ($\mathcal{A}_{\text{random}}$) and display empirical cdfs of these distributions, along with bootstrapped pdfs of the means of these distributions, in Fig. 6. Elite individuals (the top $k = 5$ performers on the validation dataset) have the greatest maximum absolute difference (Kolmogorov-Smirnov statistic) between the empirical cdf of their profit and the other cdfs ($D(\mathcal{A}_{\text{elite}}, \mathcal{A}_{\text{random}}) = 0.4488$, $D(\mathcal{A}_{\text{elite}}, \mathcal{A}_{10}) = 0.3809$) while the maximum distance between the empirical cdf of random neural network profit and all evolved neural network profit was smaller, but still significantly greater than zero ($D(\mathcal{A}_{10}, \mathcal{A}_{\text{random}}) = 0.0956$), as demonstrated in panel A. Though the $\mathcal{A}_{10}$ and $\mathcal{A}_{\text{random}}$ profit distributions are far more similar to each other than they are to that of $\mathcal{A}_{\text{elite}}$, they differ significantly in their tails: the $\mathcal{A}_{10}$ distribution has a higher likelihood of observing larger losses (though the magnitude of these losses are still severely damped by the risk management routines detailed in Sec. 2.4) and larger gains than does the $\mathcal{A}_{\text{random}}$ distribution. It is possible this occurred because many agents in $\mathcal{A}_{10}$ had high fitness in their particular realization of the ABM with agent concentration vector α, but α did not resemble the (effectively unobservable) makeup of agents that generated the real spot price time series. The mean profit of $\mathcal{A}_{10}$ on validation data was higher than that of $\mathcal{A}_{\text{random}}$ on test data, though estimates of this mean (displayed in panel B of Fig. 6) have greater dispersion that estimates of the mean of $\mathcal{A}_{10}$ profit. It is likely (probability ≈ 0.7617) that the true mean of the $\mathcal{A}_{10}$ profit distribution is greater than the mean of the $\mathcal{A}_{\text{random}}$ profit distribution. However, distributions of estimated mean for both sets of agents do contain zero (though the estimated probabilities that the mean is greater than zero are 0.7863 for $\mathcal{A}_{\text{random}}$ and 0.9241 for $\mathcal{A}_{10}$).

Crucially, mean $\mathcal{A}_{\text{elite}}$ profit on test data is very far from zero (approximately 0.267 USD per trading episode) and the estimated distribution of mean profit is bounded away from zero. We discuss these results further in the supplementary information.

4 DISCUSSION AND CONCLUSION

We construct a method to develop *ab initio* trading algorithms using an agent-based model of a financial market. We subject expressive agents to evolutionary pressure, using profit generated in the agent-based model as the fitness function. We then backtest high-performing agents on real financial asset price data (spot foreign exchange rates). We further tested "elite" evolved agents—agents that performed well on validation data (EUR/USD and GBP/USD from 2010 to 2015)—on test data, the same currency pairs but during the time interval 2016 through 2019. We find that it is possible for evolved trading algorithms to make significantly positive backtesting profit for extended periods of time during varying market conditions, even though these evolved algorithms had never experienced real financial data during the process of evolution. This result provides evidence that a paradigm shift in the design of automated trading algorithms— from prediction of future market states to, instead, closely modeling the underlying mechanisms and agents of which the market is composed—may be both feasible and profitable.

Though this result is promising, it is important to note some shortcomings of our research and avenues for further exploration. First, and most importantly, we have not actually tested the performance of the "elite" evolved agents in a real market, but only backtested them on real market data. The difference between these actions is profound; the ultimate expression of confidence in a trading strategy is to use it with one's own capital and we have not yet done this [54]. Though the elite agents are consistently profitable when backtested, this does not guarantee that they will be

[5] $16.2 \text{ days} \approx \frac{10^6 \text{ s}}{60 \text{ s/m} \times 60 \text{ m/h} \times 24 \text{ h/trading day} \times \frac{5 \text{ trading days}}{7 \text{ days}}}$

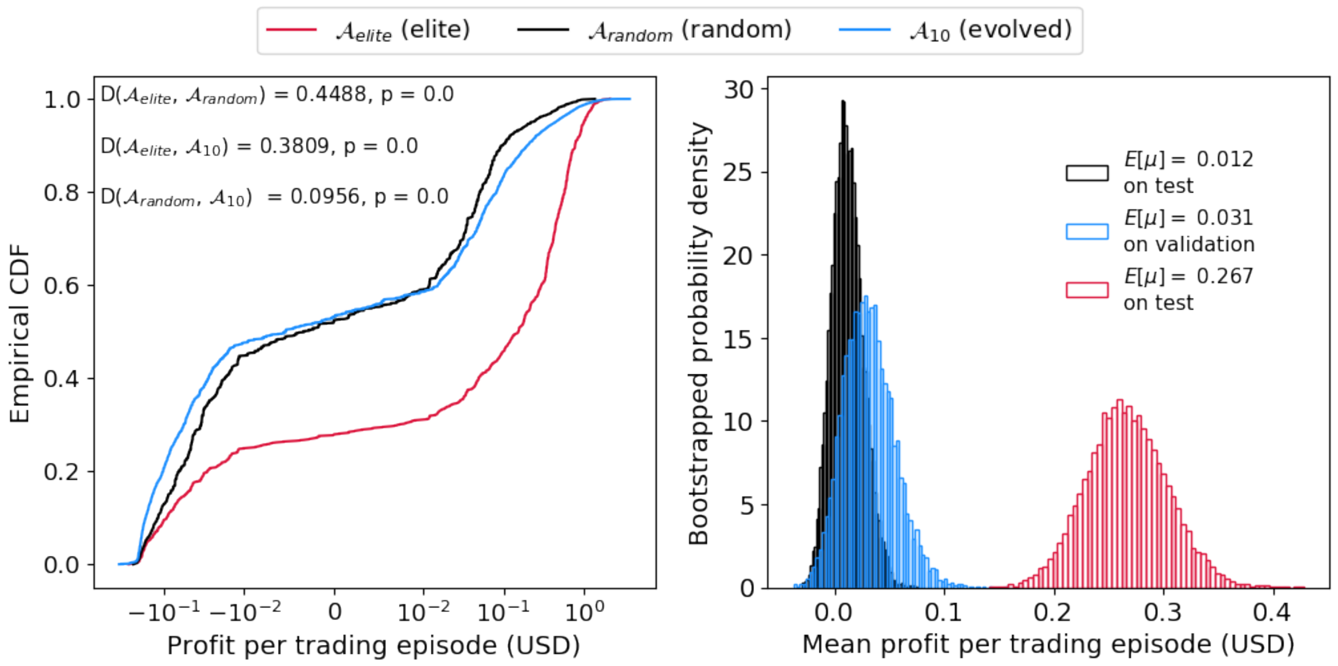

Figure 6: The profit distributions of all evolved neural networks ($\mathcal{A}_{10}$), random neural networks ($\mathcal{A}_{\text{random}}$), and elite individuals ($\mathcal{A}_{\text{elite}}$) when evaluated on real FX spot rate data differ significantly, as we demonstrate in panel A. In panel B, we demonstrate that elite evolved neural network trading strategies have significantly higher mean profits on test data than do random neural network strategies or the set of all evolved strategies evaluated on validation data. (The separation between validation and test data is irrelevant for the set of all evolved neural networks, as these networks evolved in the agent-based model, not through evaluation on real data.) The data do not appear to have a diverging second moment; we do not concern ourselves with issues that arise with bootstrapping in distributions with tail exponent $\alpha < 2$ [4].

profitable when used to trade "live" in a real financial market. If elite agents generated according to the methodology laid out in Secs. 2.3 and 2.4 are profitable when used to trade spot contracts in foreign exchange markets, we will be more confident in stating that this methodology is a robust method of generating *ab initio* trading algorithms.

Second, our implementations of multiple components of our methodology were intended as proofs-of-concept; four-layer feed-forward neural networks are decidedly not at the cutting-edge of neural network design [8]. Future work could focus on improving the expressiveness and realism of agents used in the agent-based model, modifying the evolvable neural networks to use a recurrent architecture, using different order types in the matching engine (and hence increasing dimensionality of the neural networks' action space), and in general attempting to more closely match the composition of the agent-based model with the structure of modern-day securities markets (in particular, spot FX rate markets). We could also subject other types of agents to moderate evolutionary pressure in order to simulate the knowledge dissemination that occurs in communities of relatively static strategies [16].

Finally, there is substantial room for more analysis of the free parameters in our agent-based model—for example, tuning the parameters of the tournament selection adaptively so that later generations do not evolve to simply exploit the structure of the agent-based model but rather continue to explore novel trading strategies. More generally, we should improve the design of the mechanism by which we select high-performing individuals from the model to be backtested on real data. We have used only a heuristic measure—the apparent stabilization of the MSD exponent and decreasing marginal log profit—as indicators as from which generation we should select, and what follows this is essentially rejection sampling from the space of agents that are high-performers in the agent-based model implemented by evaluation of these agents on validation data. We believe that there are probably better ways to implement this step.

ACKNOWLEDGMENTS

The authors are grateful for helpful conversations with Josh Bongard, Christopher Danforth, Peter Dodds, Sam Kriegman, Josh Minot, and Colin Van Oort. The authors are also thankful for the financial and physical support of the Massachusetts Mutual Life Insurance Company.

REFERENCES

[1] Natalia Akchurina. 2009. Multiagent reinforcement learning: algorithm converging to nash equilibrium in general-sum discounted stochastic games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. Citeseer, 725–732.

[2] Eric M Aldrich and K López Vargas. 2018. Experiments in high-frequency trading: Testing the frequent batch auction. *Experimental Economics* (2018).

[3] Jaakko Aspara and Arvid OI Hoffmann. 2015. Cut your losses and let your profits run: How shifting feelings of personal responsibility reverses the disposition effect. *Journal of Behavioral and Experimental Finance* 8 (2015), 18–24.

[4] KB Athreya et al. 1987. Bootstrap of the mean in the infinite variance case. *The Annals of Statistics* 15, 2 (1987), 724–731.

[5] Mark P Austin, Graham Bates, Michael AH Dempster 3, Vasco Leemans, and Stacy N Williams. 2004. Adaptive systems for foreign exchange trading. *Quantitative Finance* 4, 4 (2004), 37–45.

[6] Lawrence M Ausubel. 2004. An efficient ascending-bid auction for multiple objects. *American Economic Review* 94, 5 (2004), 1452–1475.

[7] Swaminathan G Badrinath and Sunil Wahal. 2002. Momentum trading by institutions. *The Journal of Finance* 57, 6 (2002), 2449–2478.

[8] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167* (2016).

[9] Marco Bartolozzi and Anthony William Thomas. 2004. Stochastic cellular automata model for stock market dynamics. *Physical review E* 69, 4 (2004), 046112.

[10] Fischer Black and Myron Scholes. 1973. The pricing of options and corporate liabilities. *Journal of political economy* 81, 3 (1973), 637–654.

[11] Tobias Blickle and Lothar Thiele. 1996. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation* 4, 4 (1996), 361–394.

[12] Eric Budish, Peter Cramton, and John Shim. 2014. Implementation details for frequent batch auctions: Slowing down markets to the blink of an eye. *American Economic Review* 104, 5 (2014), 418–24.

[13] Eric Budish, Peter Cramton, and John Shim. 2015. The high-frequency trading arms race: Frequent batch auctions as a market design response. *The Quarterly Journal of Economics* 130, 4 (2015), 1547–1621.

[14] Charles Cao, Oliver Hansch, and Xiaoxin Wang. 2009. The information content of an open limit-order book. *Journal of Futures Markets: Futures, Options, and Other Derivative Products* 29, 1 (2009), 16–41.

[15] Louis KC Chan, Narasimhan Jegadeesh, and Josef Lakonishok. 1996. Momentum strategies. *The Journal of Finance* 51, 5 (1996), 1681–1713.

[16] Yeganeh Charband and Nima Jafari Navimipour. 2016. Online knowledge sharing mechanisms: a systematic review of the state of the art literature and recommendations for future research. *Information Systems Frontiers* 18, 6 (2016), 1131–1151.

[17] Shu-Heng Chen and Chia-Hsuan Yeh. 2001. Evolving traders and the business school with genetic programming: A new architecture of the agent-based artificial stock market. *Journal of Economic Dynamics and Control* 25, 3-4 (2001), 363–393.

[18] Shu-Heng Chen, Ren-Jie Zeng, and Tina Yu. 2009. Co-Evolving Trading Strategies to Analyze Bounded Rationality in Double Auction Markets. In *Genetic programming theory and practice VI*. Springer, 1–19.

[19] Silvano Cincotti, Sergio M Focardi, Michele Marchesi, and Marco Raberto. 2003. Who wins? Study of long-run trader survival in an artificial stock market. *Physica A: Statistical Mechanics and its Applications* 324, 1-2 (2003), 227–233.

[20] Carol Clark et al. 2010. Controlling risk in a lightning-speed trading environment. *Chicago Fed Letter* 272 (2010).

[21] Carole Comerton-Forde and James Rydge. 2006. The influence of call auction algorithm rules on market efficiency. *Journal of Financial Markets* 9, 2 (2006), 199–222.

[22] Rama Cont. 2007. Volatility clustering in financial markets: empirical facts and agent-based models. In *Long memory in economics*. Springer, 289–309.

[23] AAG Cortines and R Riera. 2007. Non-extensive behavior of a stock market index at microscopic time scales. *Physica A: Statistical Mechanics and its Applications* 377, 1 (2007), 181–192.

[24] Sanmay Das. 2008. The effects of market-making on price dynamics. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 887–894.

[25] Clive Davidson. 2012. A dark Knight for algos. *Risk* 25, 9 (2012), 32.

[26] Werner FM De Bondt and Richard H Thaler. 1989. Anomalies: A mean-reverting walk down Wall Street. *Journal of Economic Perspectives* 3, 1 (1989), 189–202.

[27] Michael AH Dempster and Chris M Jones. 2001. A real-time adaptive trading system using genetic programming. *Quantitative Finance* 1, 4 (2001), 397–413.

[28] Sandhya Devi. 2017. Financial market dynamics: superdiffusive or not? *Journal of Statistical Mechanics: Theory and Experiment* 2017, 8 (2017), 083207.

[29] Bernard S Donefer. 2010. Algos gone wild: Risk in the world of automated trading strategies. *The Journal of Trading* 5, 2 (2010), 31–34.

[30] J Doyne Farmer, Paolo Patelli, and Ilija I Zovko. 2005. The predictive power of zero intelligence in financial markets. *Proceedings of the National Academy of Sciences* 102, 6 (2005), 2254–2259.

[31] David Garcia and Frank Schweitzer. 2015. Social signals and algorithmic trading of Bitcoin. *Royal Society open science* 2, 9 (2015), 150288.

[32] Dhananjay K Gode and Shyam Sunder. 1993. Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *Journal of political economy* 101, 1 (1993), 119–137.

[33] Anton Golub, James B Glattfelder, and Richard B Olsen. 2018. The alpha engine: designing an automated trading algorithm. In *High-Performance Computing in Finance*. Chapman and Hall/CRC, 49–76.

[34] Amy Greenwald. 2003. The 2002 trading agent competition: An overview of agent strategies. *AI Magazine* 24, 1 (2003), 83–83.

[35] Lawrence E Harris and Venkatesh Panchapagesan. 2005. The information content of the limit order book: evidence from NYSE specialist trading decisions. *Journal of Financial Markets* 8, 1 (2005), 25–67.

[36] Johannes Heinrich, Marc Lanctot, and David Silver. 2015. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning*. 805–813.

[37] Jimmy E Hilliard, Jeff Madura, and Alan L Tucker. 1991. Currency option pricing with stochastic domestic and foreign interest rates. *Journal of Financial and Quantitative Analysis* 26, 2 (1991), 139–151.

[38] Narasimhan Jegadeesh and Sheridan Titman. 1993. Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of finance* 48, 1 (1993), 65–91.

[39] Narasimhan Jegadeesh and Sheridan Titman. 2001. Profitability of momentum strategies: An evaluation of alternative explanations. *The Journal of finance* 56, 2 (2001), 699–720.

[40] Andrei Kirilenko, Albert S Kyle, Mehrdad Samadi, and Tugkan Tuzun. 2017. The Flash Crash: High-frequency trading in an electronic market. *The Journal of Finance* 72, 3 (2017), 967–998.

[41] Andrei A Kirilenko and Andrew W Lo. 2013. Moore's law versus murphy's law: Algorithmic trading and its discontents. *Journal of Economic Perspectives* 27, 2 (2013), 51–72.

[42] Robert Kissell and Roberto Malamut. 2005. Understanding the profit and loss distribution of trading algorithms. *Trading* 2005, 1 (2005), 41–49.

[43] Blake LeBaron. 2002. Building the Santa Fe artificial stock market. *Physica A* (2002).

[44] Blake LeBaron, W Brian Arthur, and Richard Palmer. 1999. Time series properties of an artificial stock market. *Journal of Economic Dynamics and control* 23, 9-10 (1999), 1487–1516.

[45] Serafin Martinez-Jaramillo and Edward PK Tsang. 2009. An heterogeneous, endogenous and coevolutionary GP-based financial market. *IEEE Transactions on Evolutionary Computation* 13, 1 (2009), 33–55.

[46] R Preston McAfee and John McMillan. 1987. Auctions and bidding. *Journal of economic literature* 25, 2 (1987), 699–738.

[47] Fredrick Michael and MD Johnson. 2003. Financial market dynamics. *Physica A: Statistical Mechanics and its Applications* 320 (2003), 525–534.

[48] Paul R Milgrom and Robert J Weber. 1982. A theory of auctions and competitive bidding. *Econometrica: Journal of the Econometric Society* (1982), 1089–1122.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[50] Maureen O'Hara and George S Oldfield. 1986. The microeconomics of market making. *Journal of Financial and Quantitative analysis* 21, 4 (1986), 361–376.

[51] RG Palmer, W Brian Arthur, John H Holland, and Blake LeBaron. 1999. An artificial stock market. *Artificial Life and Robotics* 3, 1 (1999), 27–31.

[52] Adam Ponzi and Yoji Aizawa. 2000. Evolutionary financial market models. *Physica A: Statistical Mechanics and its Applications* 287, 3-4 (2000), 507–523.

[53] Thomas Philip Runarsson and Simon M Lucas. 2005. Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *IEEE Transactions on Evolutionary Computation* 9, 6 (2005), 628–640.

[54] Constantine Sandis and Nassim Taleb. 2014. The Skin in the Game Heuristic for Protection Against Tail Events. *Review of Behavioral Economics* (2014).

[55] Gheorghe Cosmin Silaghi and Valentin Robu. 2005. An agent strategy for automated stock market trading combining price and order book information. In *2005 ICSC Congress on Computational Intelligence Methods and Applications*. IEEE, 4–pp.

[56] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[57] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.

[58] Didier Sornette and Susanne von der Becke. 2011. Crashes and High Frequency Trading: An evaluation of risks posed by high-speed algorithmic trading. *The Future of Computer Trading in Financial Markets* (2011).

[59] Brian F Tivnan, David Rushing Dewhurst, Colin M Van Oort, IV Ring, H John, Tyler J Gray, Brendan F Tivnan, Matthew TK Koehler, Matthew T McMahon, David Slater, et al. 2019. Fragmentation and inefficiencies in US equity markets: Evidence from the Dow 30. *arXiv preprint arXiv:1902.04690* (2019).

[60] Hanqin Zhang and Qing Zhang. 2008. Trading a mean-reverting asset: Buy low and sell high. *Automatica* 44, 6 (2008), 1511–1518.

Supplementary information for "Evolving *ab initio* trading strategies in heterogeneous environments"

Algorithmic details of agents

All agents (except for neural network agents) submit a number of shares that is Poisson distributed with mean value $E[N] = 100$ by default.

- Zero intelligence (zi): submit bid with probability $s_t \sim \text{Bernoulli}(p_{\text{bid}})$. We set $p_{\text{bid}} = \frac{1}{2}$. Order price is distributed uniformly around the last equilibrium price from the matching engine, $X_t^{(o)} = X_{t-1} + \nu u_t$, where $u_t \sim \mathcal{U}(-1, 1)$ and $\nu > 0$ is the so-called "micro-volatility" preference (denoted below by `self.vol_pref`) of the agent [?]. We have set $\nu \sim \text{Exponential}(\beta = 10)$ in this study. This random variable is set at runtime and remains constant for the life of the agent.

- Zero intelligence plus (zip): the same as a zero-intelligence agent except they submit no order price; their orders are market orders.

- Momentum (mo): these agents implement a very simple momentum algorithm based on the change in price between the current price and the last observed price:

```
dp = self.last - p  # p is the price; self.last is the last observed price
if dp > 0:
    side = 'ask'
    price = p + self.dx  # self.dx is a small price increment
elif dp < 0:
    side = 'bid'
    price = p - self.dx
else:
    side = np.random.choice(['ask', 'bid'])
    price = p
```

  We set the price increment $dx = 0.05$, but this is obviously arbitrary. In particular, one could set this to be a random variable that takes other information about market state as parameters.

- Fundamental value: these agents have a "true valuation" estimate of what they think the traded asset is actually worth. If the price of the asset is below this value, they submit a bid, while if it is above this value, they submit an ask.

```
if p == orders.NaP:  # how we denote no price information returned by matching engine
    return []
# mean_price is the agent's true-valuation estimate
# price_tolerance is the agent's estimate of error around the true mean price
# that they are willing to accept
elif p > self.mean_price + self.price_tolerance:
    side = 'ask'
    price = p + self.dx + self.vol_pref * np.random.random() - self.vol_pref / 2  #
        vol_pref is nu from above
elif p < self.mean_price - self.price_tolerance:
    side = 'bid'
    price = p - self.dx + self.vol_pref * np.random.random() - self.vol_pref / 2
else:
    return []
```

  As with any other agent with a `vol_pref` attribute, this algorithm can be made deterministic by setting `self.vol_pref` $= 0$.

- Mean reversion: this agent anticipates a return to some rolling mean. If the price is higher than the rolling mean, the agent submits an ask order. If the price is lower than the rolling mean, it submits a bid order.

```
self.prices[self.current_ind] = p  # a list of recent prices
# self.window is number of prices that are stored
self.current_ind = (self.current_ind + 1) % self.window

self.mean_price = self.prices[self.prices > 0].mean()
if p == orders.NaP:
    return []
elif p > self.mean_price + self.price_tolerance:
    side = 'ask'
    price = p - self.vol_pref * np.random.random()
elif p < self.mean_price - self.price_tolerance:
    side = 'bid'
    price = p + self.vol_pref * np.random.random()
else:
    return []
```

We set `self.window` = 50 by default.

- Market making: this agent provides liquidity on both sides of a limit order book in an attempt to collect small profits that result from other market participants crossing the spread. Like the mean reverting agent, this agent keeps a rolling list of recent prices to which it refers when calculating likelihood of price movements.

```
self.prices[self.current_ind] = self.engines[eid].eq  # interfacing with one of
   possibly many matching engines
self.current_ind = (self.current_ind + 1) % self.window
p = self.prices[self.prices > 0].mean()

# React to inventory imbalance
# target_inventory_size is generally assumed to be a small number
divergence = (self.shares_held - self.target_inventory_size)
if divergence > self.inventory_tolerance:
    self.shift = max(0.01, self.shift - 0.01)  # shifts reference price
    self.spread += 0.01  # shifts how far on either side of the equilibrium the agent
        will place orders
elif divergence < -self.inventory_tolerance:
    self.shift += 0.01  # shift price up
    self.spread += 0.01
else:
    self.shift = 0.
    self.spread = max(0.01, self.spread - 0.01)
```

Unlike the other agents described, the market-making agent then submits two orders instead of only one.

```
buy_order = orders.Order(
    self.uid,
    f'{self.uid}-{self.order_number}',
    'bid',
    max(int(self.shares - divergence), 10),  # adaptively calculate number of shares
       to lower inventory imbalance
    np.round(p - self.spread + self.shift, 2),
    time_in_force=0,
)
self.order_number += 1

sell_order = orders.Order(
    self.uid,
    f'{self.uid}-{self.order_number}',
    'ask', max(int(self.shares + divergence), 10),
        np.round(p + self.spread + self.shift, 2),
        time_in_force=0,
    )
    self.order_number += 1
```

Marginalization procedure

We provide more details on how we attempt to marginalize over unobserved market variables using analogues of those variables generated by the agent-based model (ABM). Recall from the main paper that we can view the ABM as a stochastic function $\mathcal{G}(\alpha, \mathcal{M})$ that, given an agent parameter vector α and evolutionary mechanism $\mathcal{M}$ (which may be the identity mechanism, i.e. no selection or mutation), generates orderbooks $D_b(x,t)|\alpha$, $D_a(x,t)|\alpha$ and price time series $X_t|\alpha$. Our first step is simply to jointly obtain many orderbooks and price time series by calling $\mathcal{G}(\alpha, \mathcal{M})$ many times for a variety of different α. We then have random fields of orderbooks $\mathcal{D}_b = \{D_b(x,t)|\alpha\}_\alpha$, $\mathcal{D}_a = \{D_a(x,t)|\alpha\}_\alpha$ and price time series $\mathcal{X} = \{X_t|\alpha\}_\alpha$ indexed by the vectors α that we draw from some joint distribution $p(\alpha)$. In our implementation, we just set $p(\alpha)$ to a multinomial distribution with equal probabilities of picking each agent type.

We called $\mathcal{G}(\alpha, \mathrm{id})$ once for each saved evolved neural network and added the evolved neural network to the simulation. Given these random fields of orderbooks and price time series, we create the price difference $\Delta X = X_t - X_{t-1}$ and the total resting order volume time series calculated from the sequence of orderbooks, $\Delta\hat{V}^{(b)}$ and $\Delta\hat{V}^{(a)}$, defined analogously. The change in price does have an observable real market analogue—the actual change in price of the spot asset—while the change in resting order volume does not have an observable real market analogue, since we did not have access to real market orderbook data. We store ΔX in a ball tree using the ℓ_1 distance metric. Given an observed ΔX^* from real market data, we then query the ball tree for the indices $i_1, ..., i_k$ of the nearest k neighbors of ΔX^*, $\Delta X_{i_1}, ..., \Delta X_{i_k}$ and then extract $\Delta\hat{V}^{(b)}_{i_1}, ..., \Delta\hat{V}^{(b)}_{i_k}$ and $\Delta\hat{V}^{(a)}_{i_1}, ..., \Delta\hat{V}^{(a)}_{i_k}$. We then approximate the market $(\Delta V^{(b)}, \Delta V^{(a)})|\Delta X^*$ by the approximation to the conditional expectation given by the result of the nearest-neighbors query:

$$\Delta V^{(b)}|\Delta X^* \approx \frac{1}{k}\sum_{j=1}^{k} \Delta\hat{V}^{(b)}_{i_j}, \qquad \Delta V^{(a)}|\Delta X^* \approx \frac{1}{k}\sum_{j=1}^{k} \Delta\hat{V}^{(a)}_{i_j}. \tag{1}$$

The validity of this procedure rests on the degree to which the mechanics and dynamics of the ABM simulate the true mechanics and dynamics of the real asset market.

Risk management routines

We implemented some simple risk management routines to halt the losses of poorly-performing algorithms. These routines come in three variants: two versions of the traders' adage "cut your losses but let your winners run," and one version that seeks to limit total leverage (net open position of spot contracts).

- "Cut losses" algorithm operating on price levels: halts trading if total profit is below a certain level.

```
    def cut_losses(
        profit_arr,
        lower_limit=-0.5,
        method='absolute',
        roll_ind=100,
        ):
    """Implements the level-based adage "cut your losses and let your winners run."

    If the level of profit is below a certain set level, halt trading. Otherwise,
        keep going.

    :param profit_arr: array of profits so far
    :type profit_arr: index-able
    :param lower_limit: the level of profit below which trading should halt.
    :param method: one of `absolute` or `rolling`, how to calculate the maximum
        profit setpoint
    :type lower_limit: `float`

    :returns : `bool`, whether or not trading should halt
    """
    if (method == 'absolute') or (len(profit_arr) <= roll_ind):
        if profit_arr[-1] <= lower_limit:
            return True
        return False

    elif method == 'rolling':
        max_profit = np.max( profit_arr[-roll_ind:] )
        if profit_arr[-1] - lower_limit < max_profit:
            return True
        return False

    else:  # they didn't specify anything, better to halt rather than propagate their
         errors
        return True
```

If `method == 'rolling'` and `roll_ind` $= 0$, this is equivalent to halting trading at time t if π_t is less than $\max_{t'=1,\ldots,t} \pi_{t'} -$ `lower_limit`; we used these parameter settings during our experiments.

- "Cut losses" algorithm operating on changes in price level: halts trading if $\Delta\pi$ is less than some specified cutoff.

```
    def cut_losses_delta(
        profit_arr,
        lower_limit=-0.5,
        ):
    """Implements the flow-based adage "cut your losses and let your winners run."

    If the change in profit is below a certain set level, halt trading. Otherwise,
        keep going.

    :param profit_arr: array of profits so far
    :type profit_arr: index-able
    :param lower_limit: the level of delta profit below which trading should halt.
    :type lower_limit: `float`

    :returns : `bool`, whether or not trading should halt

    """
    if (len(profit_arr) >= 2) and (profit_arr[-1] - profit_arr[-2] <= lower_limit):
        return True
    return False
```

- Leverage limit: if the total number of spot contracts (long or short, i.e., positive or negative) is above a set threshold, halts trading.

```
    def limit_leverage(
        shares_arr,
        max_shares=150,
        ):
    if np.abs( shares_arr[-1] ) >= max_shares:
        return True
    return False
```

The existence of a risk management routine supervising an algorithm's execution will, in general, change the algorithm's profit distribution. As an example, suppose that a trading algorithm had zero average profit $E[\pi] = \int_{-\infty}^{\infty} \pi p(\pi)\ d\pi = 0$ when unsupervised by a risk management algorithm. If the risk management algorithm were "perfect" in the sense that it could guarantee limit loss to no less than $-\pi^*$ for $\pi^* > 0$, then it is the case that the trading algorithm would have positive expected profit, since expected profit is then given by

$$\int_{-\infty}^{\infty} \max\{-\pi^*, \pi\}\ p(\pi)\ d\pi \geq \int_{-\infty}^{\infty} \pi\ p(\pi)\ d\pi = 0. \tag{2}$$

Profitability of evolved agents

We tested the profitability of all evolved algorithms on the validation dataset, the currency pairs EUR/USD and GBP/USD from 1-1-2010 to 12-31-2015, and then tested the elite evolved algorithms—what we termed the top five most profitable algorithms on the validation dataset—on a test dataset, EUR/USD and GBP/USD from 1-1-2016 to 7-1-2019 (the date we collected this data).

Below, we display the distributions of profit by the elite evolved algorithms on the test dataset. By total profit, we mean all profit the algorithm earned over all trading episodes (defined in the main paper) in the test dataset. The probability of profit, expected profit, and maximum *a-posteriori* profit are all measured on a per-trading-episode basis. The red curve superimposed on the histogram is a maximum-likelihood estimated lognormal pdf, which fits the observed data fairly well in some cases and not well in others. We fit a lognormal pdf as, if profit accumulates via random positive and negative percentages changes, the lognormal distribution is the appropriate limiting distribution by the multiplicative CLT (after shifting the distribution by the appropriate location and scale parameters).

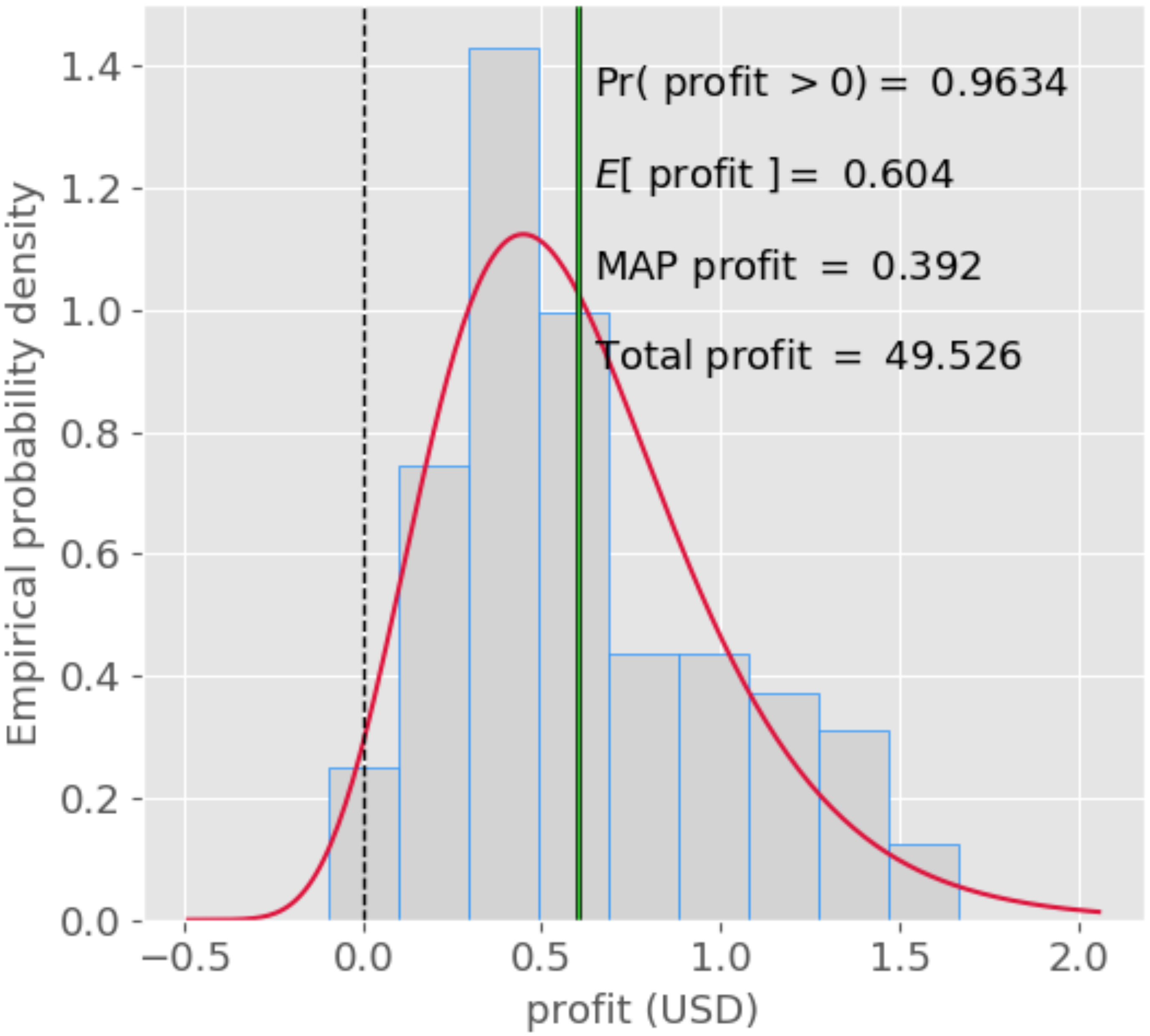

Figure 1: $g = 10$, independent simulation 21

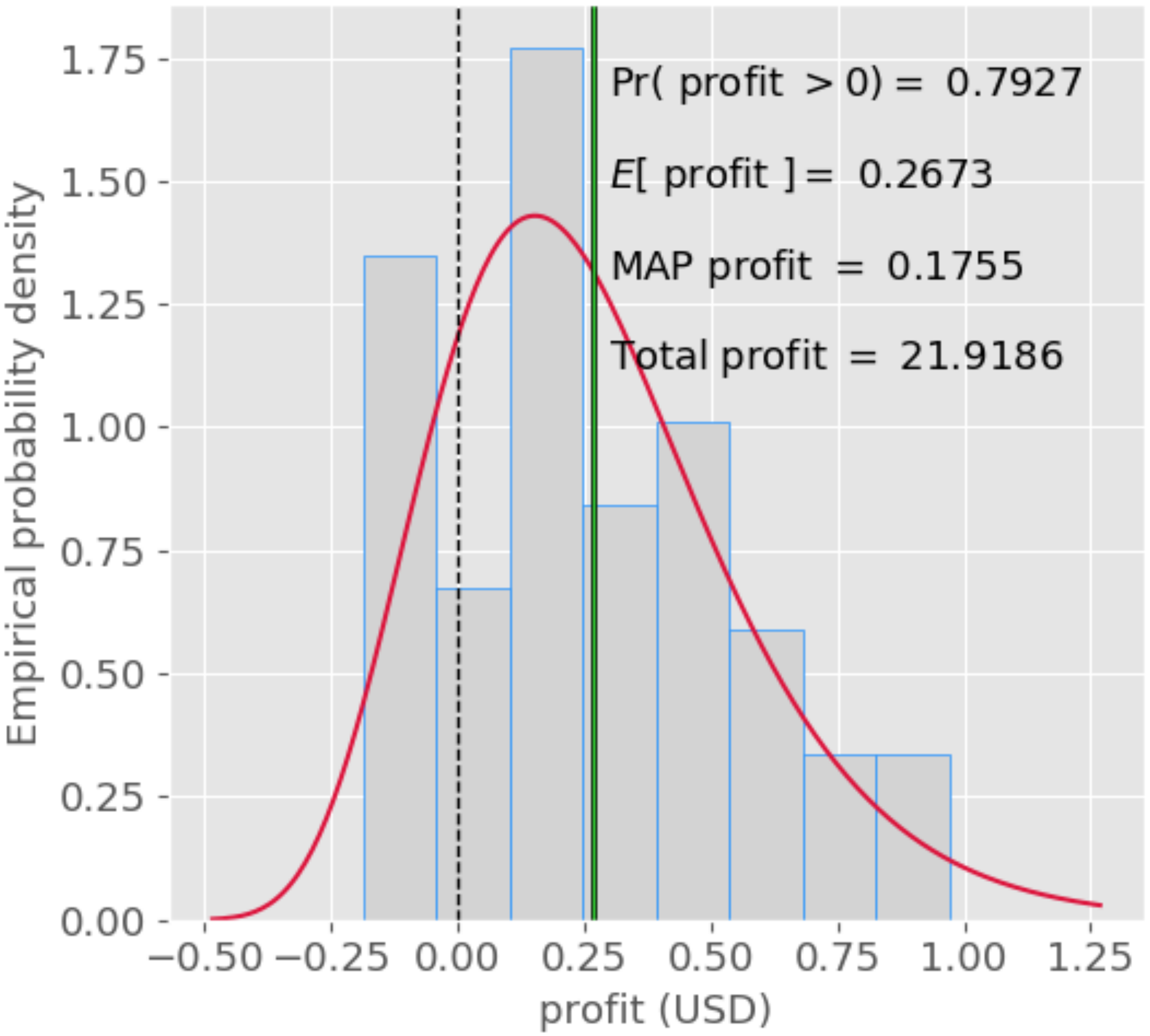

Figure 2: $g = 10$, independent simulation 1

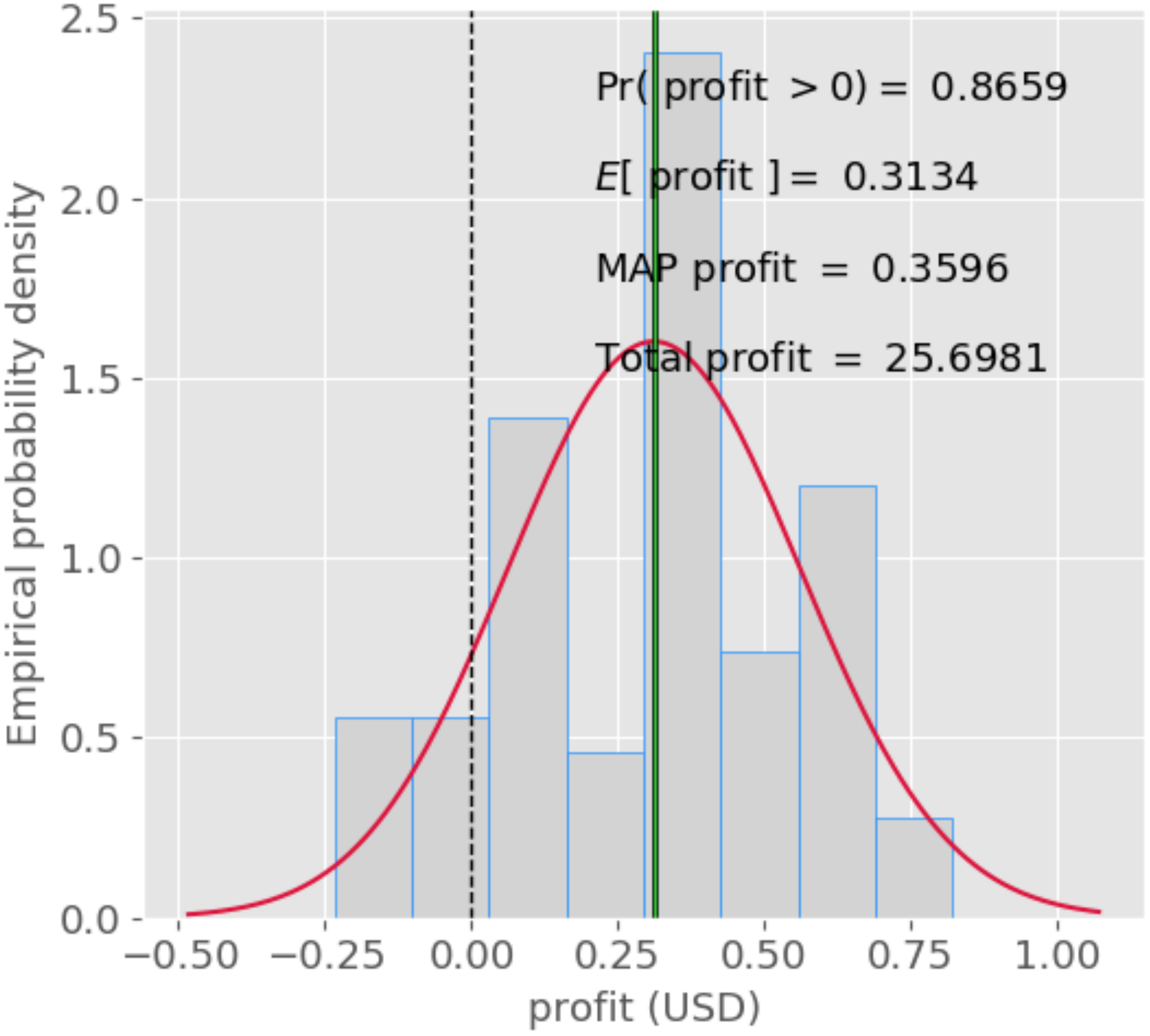

Figure 3: $g = 10$, independent simulation 93

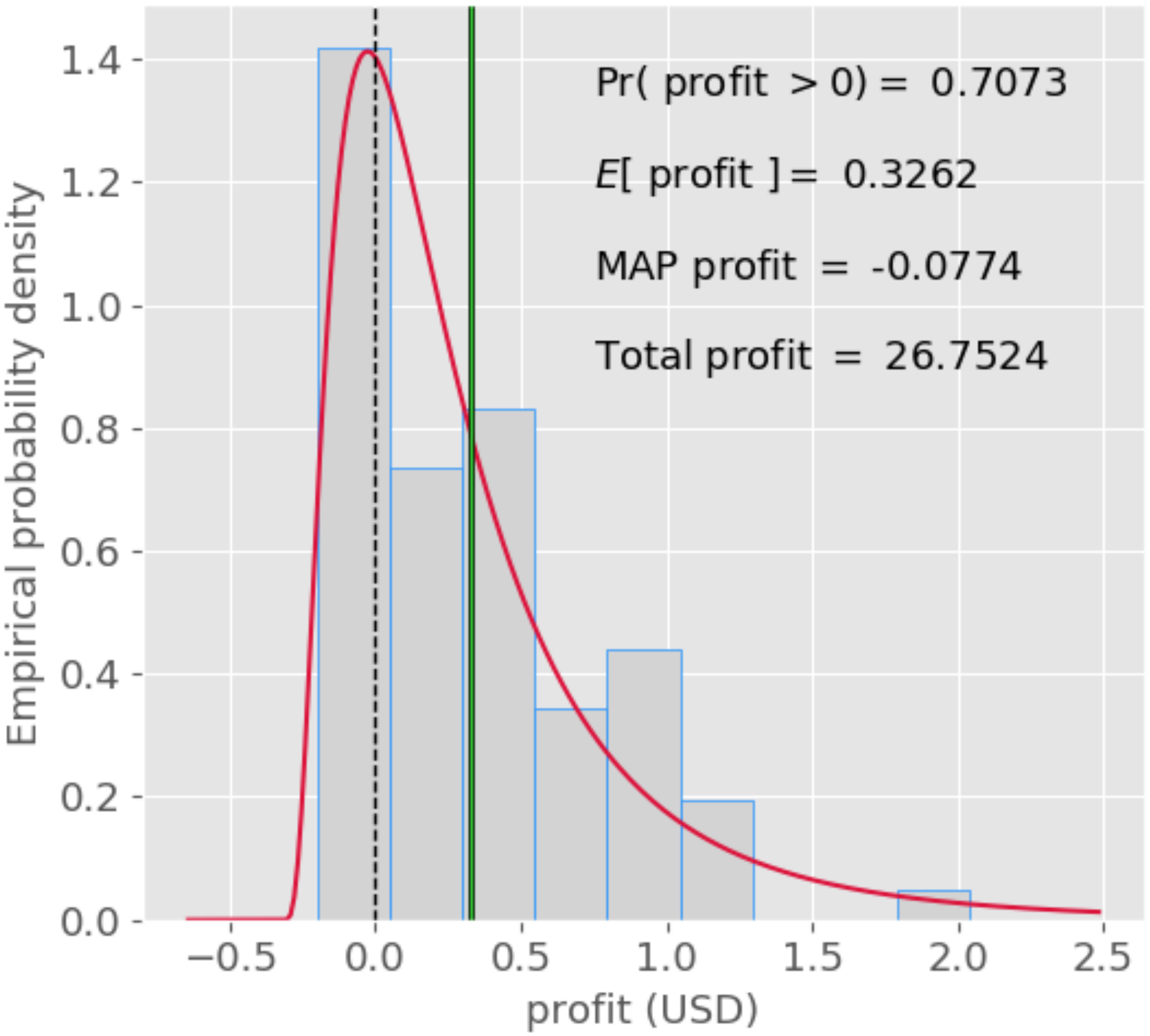

Figure 4: $g = 10$, independent simulation 79

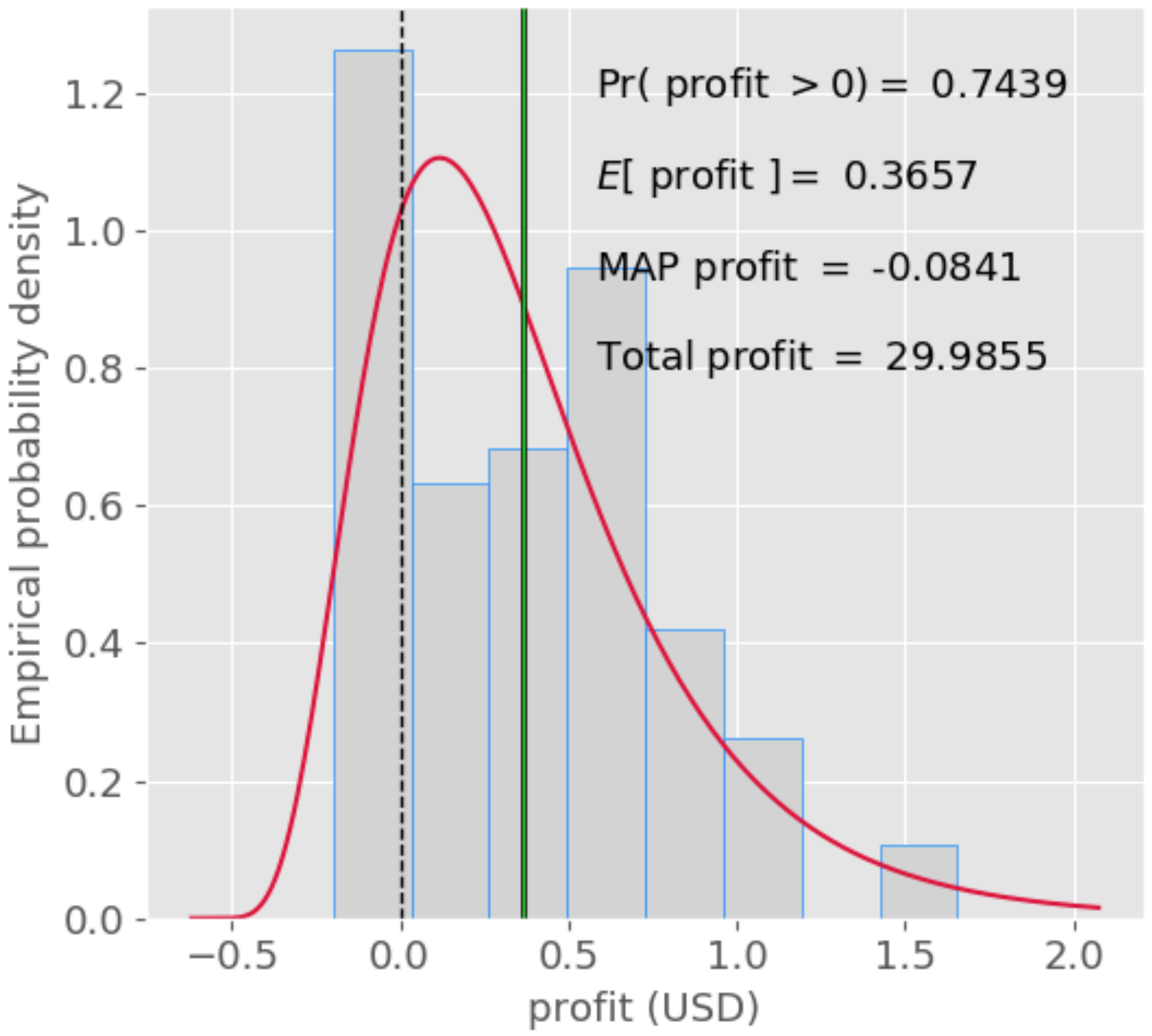

Figure 5: $g = 10$, independent simulation 38