Monte-Carlo Tests for Identification and Validation of Stochastic Agent-Based Models [Working Paper]

Christopher Zosh, Nency Dhameja, Yixin Ren, Andreas Pape*

November 14, 2025

Abstract

Agent-based models (ABMs) are increasingly used for formal estimation and inference, yet their complexity and algorithmic nature pose persistent challenges for assessing estimator properties.

This paper shows how Monte Carlo simulations (MCS) can address these challenges. We show that MCS can systematically assess whether ABM parameters are identifiable, how accuracy and precision of estimates depend on factors such as search algorithm, number of model runs, and fitness function specification. MCS can also speak to model validity. We further introduce a novel Monte Carlo test that disentangles the source of imprecision arising from model and estimation stochasticity versus sampling variation.

We demonstrate these methods using two applications: a repeated prisoner's dilemma with learning agents and a model of information diffusion on a network. In the first, we find parameters are recoverable with grid search and particle swarm optimization with sufficient runs but not with a genetic algorithm, and estimates are largely unbiased when sufficient runs are used. In the second, we see parameters are recoverable but highly sensitive to the moments used in the fitness function, and estimator behavior diverges when applied to real data, suggesting model validity issues.

Our results show that even when ABM parameters can be identified, estimator performance can be sensitive to the fitness function, search method, and model features, underscoring the need for MCS-based diagnostics before drawing substantive conclusions.

*Thank you for the research assistance of Jijee Bhattarai, Illay Gabbay, Drew Havlick, Muhammad Imam, Jack Phair, Luke Puthumana, Phil Siemers, Zhikang Tang, Case Tatro, and Weihao Zhang.

Contents

1 Introduction

While agent-based models (ABMs) and computational simulations may seem new, their history in economics (and in the social sciences at large) can be traced back to at least Schelling's segregation model (Schelling 1971). Predating the prevalence and computational power of today's computers, Schelling's model computations were performed using coins on a chessboard. Despite the simplicity of the rules introduced, an easy closed-form characterization of the dynamics of the model could not be found. To understand the implications of his simple model, Schelling performed numerous computations by hand over the board from different starting points, then aggregated and reported the results. Wielding this unconventional model and method of analysis, he illuminated how a small level of intolerance can yield a surprisingly high degree of macro-level segregation in an extremely simple system. Since then, the role such methods and models can play in understanding emergent macro-phenomenon has been a large subject of debate.

While many utilizations of computational models traditionally serve to demonstrate proof-of-principle type findings (as in Schelling's case), it is becoming increasingly common and desirable to use some ABMs directly for estimation in much the same way structural equation models are used. While the ability for ABMs to be fairly unconstrained serves as a major selling point, it also means such models diverge from conventional structures where statistical properties of the model are better understood. If direct estimation of such unconstrained models is to be taken seriously, such results should be preceded with tests establishing basic estimator properties. To accomplish this, we need a flexible test which can be applied without constraining the model structure a priori.

The primary goal of this paper is to highlight the valuable role Monte Carlo simulation (MCS) can play in addressing these challenges. MCS can systematically evaluate a range of properties for ABMs and their estimators, including parameter identifiability, and can show how estimate precision and accuracy depend on factors such as optimization algorithm choice, number of model runs, and fitness function specification. Such analysis is vital for determining the extent to which parameter estimates can be recovered and to what degree they reflect their real-world counterparts, which we demonstrate in two example applications.

Our first application explores an ABM of learning agents playing the repeated prisoner's dilemma, which we bring to lab data from Camera and Casari (2009). Before estimating the data, our MCS reveals that the model parameters can be reasonably recovered, though the choice of optimization technique matters quite a bit for achieving fairly precise estimates. Our specifications of Gridsearch and Particle Swarm Optimization far outperform our specification of the Genetic Algorithm. This imprecision is mirrored by our results when bringing the model to data.

Our second application investigates the diffusion of information over a network via the cascade model, in which agents are nodes and pass along information stochastically. This model was applied by Rand et al. (2015) to Twitter

data on mentions of Hurricane Sandy over time. Here, we model parameters that are identical across cascades. Our MCS reveals that the degree to which one diffusion parameter q (*Imitation*) can be recovered depends quite a bit on choice of moments considered in the fitness function and, to a lesser extent, the optimization technique chosen. We also find that even when applying the best performing model specifications in the MCS to real data, the technique performs both fairly poor and looks fairly different from what we saw in our Monte Carlo simulations, indicating that our version of the model may not be valid in this context.

Our findings underscore that optimization algorithm and fitness function selection can greatly impact estimate accuracy and precision. More broadly, they highlight the value of Monte Carlo simulations for evaluating when and under what conditions ABMs can produce well-identified parameters and robust estimates. And as shown in our second example, such tools can also speak to model validity concerns.

The remainder of this paper is structured as follows. Section 2 discusses the existing literature and outlines our contribution. Section 3 provides a brief overview of parameter estimation techniques for computational models and their associated confidence intervals. In Section 4, we introduce MCS as a practical tool for studying estimate accuracy, precision, and bias, and present a novel Monte Carlo test to disentangle estimation imprecision arising from the stochasticity of the ABM and search processes themselves versus that sourced from sampling variation. In Sections 5 and 6, we apply these MCS-based tests to the two example problems mentioned above and interpret their results. Finally, we conclude in Section 7.

2 Literature Review

In the econometrics simulation literature, there has been a great deal of work on developing econometric methods for estimating nonlinear models of various forms. However, many such methods require either the ability to fully describe the model with a system of (sometimes differentiable) equations (e.g., Maximum Likelihood estimation) or may require fairly restrictive assumptions about the shape of distribution of errors. For many ABMs, some of these methods are impossible or at least fairly difficult to apply, as there is often both a non-trivial role that randomness plays and some non-trivial iterative / algorithmic element to its application which cannot be described as an equation.[1] In general in econometrics literature, Monte Carlo Simulations are the go-to tool for establishing model properties with minimal functional form requirements.

The role Monte-Carlo simulation can play for guiding estimation of data-driven ABMs, however, has been fairly underexplored. For example, there is a small but growing number of texts on Agent-Based Modeling and computational modeling at large (Sayama 2015; Wilensky and Rand 2015) and in the

[1] As an exercise to demonstrate this, try describing Schelling's fairly simple segregation model (Schelling 1971) as a system of equations.

context of social systems more specifically (Miller and Page 2007; Tesfatsion and Judd 2006; Schmedders and Judd 2014; Romanowska et al. 2021). While each of these texts provides an in-depth analysis of many important features of ABMs, including laying forth design principles and exploring important past or potential future applications, none provides a thorough treatment of how one should bring such models to data or establish their estimator properties. We formalize the application of Monte-Carlo simulations in the context of ABMs, which we argue is ideally suited for such contexts.

We also rely on Simulated Method of Moments (SMM), a fairly generalizable method for estimating parameters, to define our fitness functions. SMM was first introduced in McFadden (1989) and is summarized in a number of econometric texts, including a classic text on simulation-based methods (Gourieroux and Monfort 1996). We speak to much of the underlying estimation process in greater detail in Zosh et al. (2025).

Lastly, we introduce a novel test intended to decompose estimate imprecision sources for simulation models (Test 4) in Section 4.5. To the best of our knowledge, we know of no such similar tests.

3 ABM Estimation Overview

The goal of this section is to give an abridged overview of all of the ideas and components necessary to understand the ABM estimation process which underlies our application of Monte Carlo testing. We briefly discuss estimating ABM parameters, computing their confidence intervals, and interpreting what these parameter estimates can tell us.[2]

3.1 Estimating Best-Fitting Parameters

Consider an ABM that takes some parameters θ and perhaps some input data X (from some dataset D) and returns some model output $M_A(\theta, X) \rightarrow Y_{M_A}$ which we will compare to their equivalents Y_D in the dataset D. Best-fitting parameters, then, are a set of parameters θ^* which, when used in our model, produce output Y_{M_A} which emulates the salient features of Y_D. This assessment comparing model output to data requires the analyst to define a few functions: a *summary function* $S(\cdot)$, an *aggregation function* $Agg(\cdot)$, a *fitness function* $fit(\cdot)$, and an *optimization technique* $search(\cdot)$.

A *summary function* $S(\cdot)$ produces summary statistics of output Y_{M_A} from the model M_A (using some set of parameters θ) comparable to summary statistics of data. In our case, these summary statistics will be the first n moments $\mu_1^{'}, ...\mu_n^{'}$ of each of the outputs of interest at each timestep t, as is common practice. We denote these summarized outputs from the model and data $Z(Y_{M_A}, \theta)$ and $Z(Y_D)$ respectively.

[2]For a more detailed discussion and walkthrough of the concepts discussed in this section, please see Zosh et al. (2025).

An *aggregation function* $Agg(\cdot)$ aggregates summarized output across r runs of M_A. This is necessary because we are interested in stochastic models. For such models, a single model run with our parameters in question is not sufficient to fully characterize model behavior. We denote this aggregated set of model outputs $\bar{Z}(Y_{M_A}, \theta, r)$ which, in our examples, simply calculates the average of each summary statistic generated across runs. This is given by:

$$Agg(S(M_A, \theta, X), r) = \frac{1}{r} \sum_{1}^{r} S(M_{A,r}(\theta, X)) \tag{1}$$

We denote the set of aggregated summarized output as $\bar{Z}(Y_{M_A}, \theta, r)$.

A *fitness function* $fit(\cdot)$ establishes how to compare the aggregated output of M_A, $\bar{Z}(Y_{M_A}, \theta, r)$ against the summarized output from data D, $Z(Y_D)$. We use Simulated Method of Moments, fitting some number of moments for each time step of model and data. If fitting just the first moment, as in our first example problem, then our fitness function will look like the following:

$$fit(\theta) = \frac{1}{T+1} \sum_{t=0}^{T} (\bar{\mu}_{M_A,t} - \bar{\mu}_{D,t})^2 \tag{2}$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) = \{\bar{\mu}_{M_A,t=0}, ..., \bar{\mu}_{M_A,t=T}\} \tag{3}$$

If we are utilizing more than one moment of output, as in our second example, then our fitness function using the first n moments will look as follows:

$$fit(\theta) = \frac{1}{T+1} \sum_{t=0}^{T} [\alpha_{1,t}(\bar{\mu}_{M_A,1,t} - \bar{\mu}_{D,1,t})^2 + ... + \alpha_{n,t}(\bar{\mu}_{M_A,n,t} - \bar{\mu}_{D,n,t})^2] \tag{4}$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) = \{\bar{\mu}_{M_A,1,t=0}, \bar{\mu}_{M_A,2,t=T}, ...\bar{\mu}_{M_A,n,t=0}, ..., \bar{\mu}_{M_A,n,t=T}\} \tag{5}$$

and where α_j is the weight assigned to moment j. These weights determine the relative importance of matching the first moment outputs vs. the second and so on. We assign our weights to put these relative impacts of the moments on the fitness on similar scales by using the following:

$$\alpha_{j,t} = \frac{1}{\bar{\mu}_{M_A,j,t}} \tag{6}$$

An *optimization technique* $search(\cdot)$ searches for parameters θ^* which maximizes our fitness function $fit(\cdot)$ using some search parameters δ. We explore outcomes using three different optimization technique specifications: Grid Search, the Genetic Algorithm, and Particle Swarm Optimization.

With these four functions, best-fitting parameters θ^* can be estimated by doing the following operation:

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_D)), \delta) \rightarrow \theta^* \tag{7}$$

where

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) \tag{8}$$

In words, this process searches for the set of parameters θ^* which maximizes the measure of fitness between our aggregated summarized ABM output and our summarized observations from data.

3.2 Bootstrapping Confidence Intervals

Once a method is established for estimating best-fitting parameters θ^*, a method for computing the critical values / confidence intervals for these estimates can next be appended. There are a number of ways to do this for computational models. In our examples, we generate confidence intervals C_i using block-bootstrapping. In summary, our panel data D is broken into independent blocks, which are then resampled to generate new, simulated data sets. We estimate best-fitting parameters for each of these simulated datasets to get a distribution of best-fitting parameter results, one set from each simulated dataset. Finally, we use these estimated distributions to construct confidence intervals of the desired size (in our case, 95% confidence intervals) for each parameter. These critical values are useful both for hypothesis testing and for establishing how sensitive parameter estimates are to sampling variation.

4 Establishing Properties with Monte Carlo Simulations

The focus of this section is to investigate properties of our model and estimation strategy. Since we are dealing with a model M_A which can be non-linear, highly sensitive, which may have noise enter the model non-trivially, and an estimation technique $search(\delta, \cdot)$ that itself is stochastic and does not guarantee optimal solution for an estimator we have not shown to be unbiased, one should be cautious in assuming that θ^* or our confidence intervals C are sensible.

Instead of taking on faith that this process produces sensible output, we can run a number of tests, Monte Carlo Simulation (MCS), to explore many of these unknowns. We discuss tests which can be used to gain insight about: how well our model can recover values of θ, how biased our estimates θ are, how precise our estimates are (i.e. how big our confidence intervals C are), how much of our estimate imprecision C can be attributed to stochasticity in the model and our search process (as opposed to sampling variation in the data), and how all of this changes when we alter hyper-parameters (e.g. runs r) or various search algorithm choice.

4.1 Monte-Carlo Simulation Overview

Monte-Carlo Simulations are, broadly defined, simple computational models which are used to establish properties about some distributions when a closed-form solution is not tractable. This often involves repeatedly sampling the distribution in question in some way.

In our case, we know that $search(\delta, .)$ may return different results of θ^* (see equation 7), even when fitting the same data, due to the stochastic nature of both $search(\cdot)$ and our model M_A. Thus, we can think of $search(\cdot)$, which aims to fit our model parameters to data, as returning the estimated parameters θ^* to us from some unknown underlying joint distribution over the parameter space. Our aim is to learn about this distribution of estimates.

So how can we learn about this distribution? If we knew the true parameters θ of DGP_{Data} which were used to generate our data, and if our model truly was a reasonable approximation of this DGP, then we could simply estimate our model a number of times to generate a number of estimates θ^* and see how close they are to the known true θ. Although we obviously do not know θ or the functional form of DGP_{Data}, we can do something quite similar.

Let us suppose for a moment that our model M_A is precisely the true DGP, DGP_{Data}. Next, let us choose some parameters θ for which our model is well defined. Given these two, we could then use our model M_A and the chosen parameters θ to generate a simulated dataset by recording the model output Y.

$$M_A(\theta, X) \rightarrow \hat{D} \tag{9}$$

Importantly, these data were generated using parameter values θ that are known, because we chose them. By applying our estimation technique in slightly different ways to this simulated dataset, we can learn quite a bit about our estimator. Much of the remainder of this section is devoted to exploring these variations and how such tests can help answer the questions we have introduced above in turn.

4.2 Test: Estimate Accuracy

The first and arguably most important test of the model is to establish whether or not our estimation technique can return estimates θ^* which are reasonably close to the true known values we have chosen θ.

To do this, we simply apply our estimation technique $search(\cdot)$ as we normally would to estimate model parameters θ^* for our model M_A, but using the simulated data $\hat{D}$ as if it were real data. Formally,

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\hat{D}})), \delta) \rightarrow \theta^* \tag{7}$$

Then we simply compare each value in θ and θ^* to see how off our estimates were. Such a test is a low-cost way to gain some insight into if the model has parameters which are in fact reasonably identifiable and recoverable using this

optimization technique. If your estimates are fairly off, then some experimentation (for example, by increasing runs r in $agg(\cdot)$ or trying new search parameters δ or search methods $search(\cdot)$) may be required to achieve better results. If the issue persists, then you may have model parameters which are not reasonably identifiable. We demonstrate the results for this test on an example application for a number of search algorithms and number of runs in Sections 5.2 and 6.2.

4.3 Test: Estimate Precision

To investigate the precision of our estimates, we want to establish properties about our confidence intervals C. This can be accomplished in a very similar way to what was done in the previous test, though instead of applying our process for finding the best fitting parameters, we will apply our bootstrapping method which we use to generate our confidence intervals C.

To do this, we apply the exact same bootstrapping process in section 3.2, but use our simulated dataset $\hat{D}$ instead of our real data D. Formally:

4.3.1 Construct Dataset $\tilde{\hat{D}}_k$ Using Simulated Data $\hat{D}_k$

First, we break our simulated data up into G blocks. Next, we will construct k simulated datasets. To construct a simulated dataset, we simply sample our set of blocks G times with replacement. Formally, each simulated dataset is constructed by:

$$\tilde{\hat{D}}_k = \{b^1, ...b^G | b^m \overset{\text{iid}}{\sim} U(B)\} \tag{10}$$

4.3.2 Find K Best Fits θ_k^* for Each Using the $\hat{D}_k$

Next, we find the best fitting estimates for each of the k simulated datasets $\tilde{\hat{D}}_k$,

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\tilde{\hat{D}}_k})), \delta) \rightarrow \theta_k^* \tag{11}$$

4.3.3 Construct the Critical Value(s)

Finally, we use our k best-fitting estimates to construct critical values.

$$C_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{12}$$

With these results, we can get an idea of how precise we expect our estimate to be under ideal conditions. Observing large confidence intervals (in particular, ones close to the edges of the parameter space searched on any dimension) could indicate that either more runs are needed for aggregation (reducing the model noise in fitness) or that a change in the search method $search(\cdot)$ or search parameters δ may be required as the search method as specified could be frequently settling on suboptimal local solutions. Rerunning this test after increasing runs r, altering the search method's hyper parameters δ or utilizing a different search method altogether may result in some improvement in precision.

If no such improvement is found, then it may be the case that the model's parameters cannot be reasonably distinguished between by your fitness function, and therefore are not reasonably identifiable. To get an idea of the magnitude of estimate imprecision that can be attributed simply to model and search noise, we also introduce a novel test to decompose this imprecision which is presented later (Test 4).

As we will see in our example application later, utilizing this test revealed to us that one of our three search methods which we though had reasonable hyper-parameter specifications far under-performed the other two for our application, prompting us to re-evaluate our choices of $search(\cdot)$ and δ.

4.4 Test: Estimate Bias

Bias speaks to if our method over or under estimate the parameter(s) in question on average. Formally, bias is given by:

$$Bias^i = E(\theta^{i*}) - \theta^i \tag{13}$$

We say an estimator θ^{i*} is said to be *Unbiased* when $Bias^i = 0$.

Extending the first test, we can get an estimate of Bias by simply repeating Test 1 N times (that is, finding best fitting estimates on the simulated data) using either the same data each time $\hat{D}$. Once we get these N sets of estimated parameters, for each parameter i, bias can be approximated by simply calculating the difference between the true, underlying parameter value chosen to generate the simulated data θ^i and the average of parameter estimate θ^{i*}.

$$Bias^i_m = \frac{1}{N}\sum_{j=1}^{N} \theta^{i*}_j - \theta^i \tag{14}$$

4.5 Test: Decomposing Sources of Estimate Imprecision

The purpose of this novel test is to decompose the sources of our estimate imprecision. For the purposes of this test, we can think of two types of sources of estimate imprecision. The first source of estimate imprecision comes from sampling variation. During the bootstrapping process, we introduce samples (varied through resampling) to be fit for construction of our confidence intervals C. For simulation problems, however, there is a second source of imprecision introduced by both the $search(\cdot)$ operation (which may not always find the optimum) and the aggregated model output (which may return different aggregated output each time it is prompted to run). This means our outputted Cs have a slightly different interpretation as they encode an additional source of variation. As we turn up search and aggregation parameters (as discussed in Test 2), this source of variation should in many contexts shrink to zero, leaving us with confidence intervals C which have precisely the same interpretation as in other contexts. In practice, however, turning such parameters up can be costly, and it can be

hard to know how much this second source of variation still plays a role in our confidence interval estimates.

This problem can be addressed two-fold. First, this additional variation means our confidence intervals C typically should be larger than if they were only to capture sampling variation, meaning we are already erring on the side of caution for the sake of significance testing. Second, we introduce a test to explore the degree to which variations in the aggregated model $Agg(M_A, r)$ and the search process $search(\cdot)$ play a role in estimate imprecision.

To estimate the magnitude of the role noise from non-sampling variation sources are contributing to the confidence intervals, we propose repeating the bootstrapping process (in Test 2) but removing the sample variation component. Specifically, we bootstrap without using resampled datasets. Instead, we will generate our K estimates on the exact same dataset D to observe how large our confidence intervals are when we remove the sampling variation component. Formally:

4.5.1 Find K Best Fits θ_k^* for Each Using the Original Simulated Data $\hat{D}$ k Times

Next, for each simulated, resampled dataset $\tilde{\hat{D}}_k$ we find the best fitting estimates.

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\hat{D}_k})), \delta) \rightarrow \theta_k^* \tag{15}$$

4.5.2 Construct the Critical Value(s)

Next, as before, we use our best-fitting estimates to construct critical values for each of our parameters as described in section 3.2.

$$\hat{C}_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{16}$$

Once we have the 'confidence intervals' $\hat{C}$ generated without any sampling variation (i.e., where all k of our estimates are on the same data), we can observe how much of our confidence interval size can be attributed to non-sampling variation sources. We can also compare $\hat{C}$ to the confidence intervals C normally generated using Test 2 to see the relative size comparison. As said above, in many cases, we would expect $\hat{C}$ to be able to shrink to near zero if runs r and search parameters δ are turned up infinitely high.

We will explore an example of this test and how to make such comparisons further below in our example.

5 Application: Repeated Prisoner's Dilemma with Learning Agents

5.1 Data and Background

To demonstrate our method, we first bring an ABM of simple learning agents to lab data on a version of the repeated prisoner's dilemma from Camera and Casari (2009).

The Data: In this experiment, players are grouped into one of 50 'economies' each of size four. Each round, players are paired with a random player from their economy and play a 1-shot prisoner's dilemma. At the end of each round, there is some probability p with which the economy will play another round and 1-p that the repeated game ends. Since these draws need not coincide for all economies, some economies play more rounds than others. For the purposes of our block-bootstrapping, all players in an economy will serve as our *group* g with the number of groups $G = 50$ and the group size $Size_g = 4$. Further, a *block* b_g is defined as all periods of play by players in a single economy. These 50 blocks will be the units which we bootstrap to generate our confidence intervals. For more details, see Camera and Casari (2009).

The ABM: Following directly from the experimental design, we construct an ABM M_A in which agents are initially grouped into 50 economies of size 4. Each economy plays a number of rounds corresponding precisely to their analogues in data. In each of these rounds, agents are randomly paired up with a player in their economy and play a 1-shot prisoner's dilemma with payoffs also corresponding to the experiment, given by:

Table 1: Payoff matrix for the Prisoner's Dilemma

	Cooperate	**Defect**
Cooperate	(20, 20)	(0, 25)
Defect	(25, 0)	(5, 5)

The agents in the model decide what to play each round of the game using a simplified version of the reinforcement learning model specified in Erev and Roth (1998), which was chosen for its simplicity and for its success at matching behavior in other contexts. Agents start with uniform scores (priors) about the performance of each action. Formally, the score for any potential action $\hat{a}$ is given initially as:

$$Score_{i,t=0}(\hat{a}) = Z \tag{17}$$

where Z is some constant.

Each round, when prompted to take action, each agent chooses an action with a likelihood proportional to the action's score. This likelihood of choosing any particular action is given formally as:

$$Prob_{i,t}(\hat{a}) = \frac{Score_{i,t}(\hat{a})}{\sum_{\forall a} Score_{i,t}(a)} \tag{18}$$

At the end of the round, when payoffs are awarded, each agent uses their resulting payoff to update each action's score for the subsequent period. This updating is performed as follows:

$$Score_{i,t+1}(\hat{a}) = \begin{cases} (1-R) * Score_{i,t}(\hat{a}) + \pi_{i,t} & \text{if } \hat{a} = a_{i,t} \\ (1-R) * Score_{i,t}(\hat{a}) & \text{otherwise} \end{cases} \tag{19}$$

Where $a_{i,t}$ is the action chosen by agent i this period and $\pi_{i,t}$ is the payoff received as a result.

As seen above, this decision model utilizes two parameters: the *strength of priors* Z and *recency bias* R which make up our vector of parameters θ which we fit to data. Z is the size of the score each action starts with. In general, a higher Z corresponds to a higher willingness to explore the performance of each action. R controls the relevance that agents believe past experiences have on the present problem, which spans from 0 to 1. R=0 means all past experiences are equally relevant to the current problem while R=1 means only the most recent experience is relevant.

The Structural Assumption: Before proceeding to estimation, it is important to make clear what the structural assumption made is precisely. We are assuming our agent-based model M_A (a.k.a DGP_{M_A}) is a reasonable approximation of the true data generating process DGP_D which created this data (the lab experiment and its participants). Since the structure of the part of the model pertaining to matching players and giving payoffs is fairly easy to replicate, if the structural assumption is not true, it is most likely due to a structural difference between the decision-making process agents and actual lab participants use.

On Estimating Best Fitting Parameters: If the goal is to understand behavior, then the chosen *summary function* $S(\cdot)$ should produce some summary statistic(s) of agent choices. One simple metric is to capture the portion of agents choosing 'cooperate' each period. While certainly there are other features that may also be interesting (e.g., variation across economies), the purpose of this model is to serve its role in a straightforward demonstration of the larger method. Just as the model output has been summarized, the data is also summarized in the same way.

We compute results for three different sets of runs (r=20, r=50, and r=150) of the simulation. We then apply our *aggregation function* $Agg(\cdot)$ to the summarized results from these model runs which simply averages the cooperation rate across runs for each period of play.

Next, we define our *fitness function* $fit(\cdot)$ which takes $\bar{Z}(Y_{M_A}, \theta, r)$ and $Z(Y_D)$(the summarized data and the summarized, aggregated model output) and computes mean squared error between them. This amounts to doing SMM, considering only the first moment in the fitness function as described above:

$$fit(\theta) = \frac{1}{T+1}\sum_{t=0}^{T}(\bar{\mu}_{M_A,t} - \bar{\mu}_{D,t})^2 \tag{2}$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) = \{\bar{\mu}_{M_A,t=0}, ..., \bar{\mu}_{M_A,t=T}\} \tag{3}$$

Last, an *optimization technique search*$(\cdot)$ must be chosen to search for our best fitting parameters. We report results for three such optimization techniques: grid-search, the genetic algorithm, and particle swarm optimization.

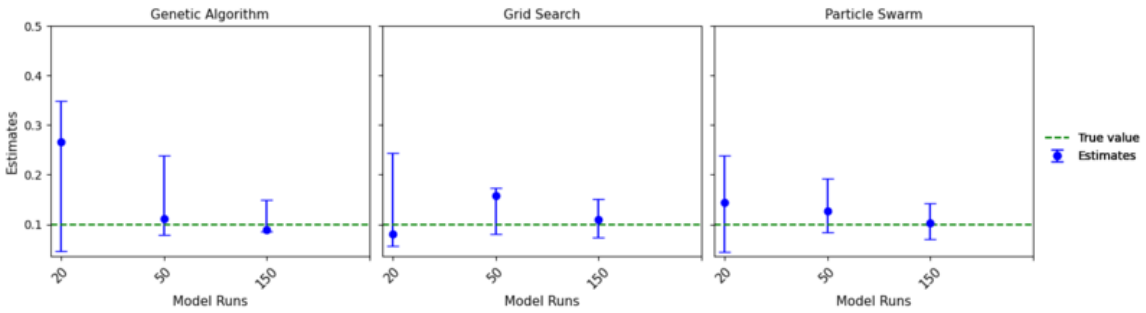
On Bootstrapping: Following the steps laid out above in Section 3.2, bootstraps can be performed by constructing a resampled dataset using our 50 blocks, then searching for parameters which best fit the data. We choose to re-sample 100 times (that is, k = 100). Then, for each parameter, we drop the outside 5% of estimates, and the remaining extremes serve as the 95% confidence intervals.

On Monte-Carlo Simulation: For Monte Carlo simulation (with the goal of establishing some metric of estimator performance), we simply perform the same estimation as we would on real data, but first must construct a 'simulated data set' as discussed in Section 4. We first choose values for each of our parameters (Z and R), then use our ABM to produce results using those parameters. In our case, we chose Z=25 and R=0.1. As previously discussed, we use the resulting unsummarized model results as if it is our lab data and proceed with the rest of the estimation process as usual. At the end, we see how well the known values for Z and R can be recovered.

5.2 Results

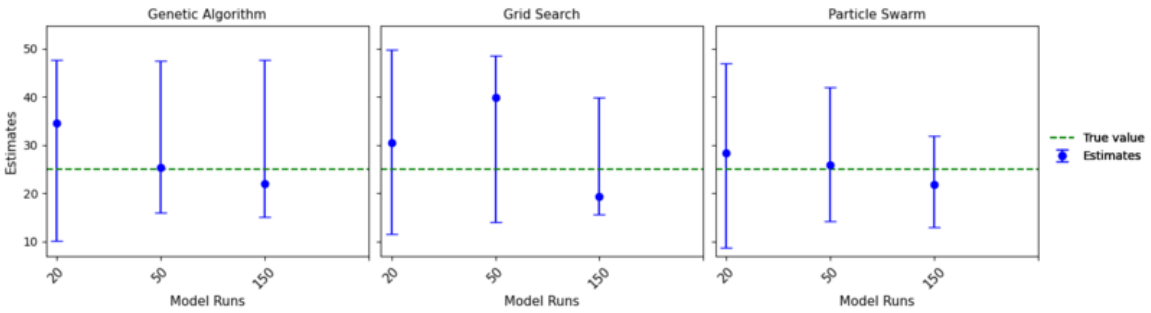
MCS Results for Best Fits and CIs: First, we show the results for both our best estimates (Test 1) and our confidence intervals (Test 2) from our Monte Carlo Simulations. We explore three levels of runs used in aggregation and three different optimization techniques, each with their own search parameters δ_j. Such exploration can give us insight on the returns additional runs have on estimate precision, on which optimization techniques seem to perform well on our problem, and if our model parameters can be identified at all.

Figure 1: Monte Carlo Simulation Results - Recency Bias (R)



In the case of our first parameter R, we can see above that all three optimization techniques perform fairly well in recovering the true value $R = 0.1$ (indicated by the green horizontal dashed line) at the highest number of runs considered. Further, all three see a non-trivial degree of confidence interval tightening from the increase in runs, with the final confidence intervals indicating a fairly high level of precision. First, this lends confidence that model parameters can be reasonably identified. Second, this demonstrates that 150 runs seems sufficient to generate fairly precise estimates of R without wasting computational resources. Finally, this seems to indicate that, in the context of estimating R, all three chosen optimization techniques seem to perform similarly well.

Figure 2: Monte Carlo Simulation Results - Prior Strength (Z)



Moving to our second parameter Z, once again it seems that with the largest number of runs, all three of the optimization techniques do fairly well at returning a best-fitting estimate close to the chosen value Z=25. It is also clear

that Particle Swarm optimization and Grid Search see some improvement in the tightness of confidence intervals with the increase in runs from 50 to 150. From these results, we can see once again that Z seems reasonably identifiable at 150 runs with these three optimization techniques. Since all of our parameters in question can be reasonably recovered, we can move forward with bringing our model to data. Second, it seems our highest number of runs (r=150) seems to help estimate precision quite a bit. Further exploration could be done to see if further gains in estimate precision can be achieved with an increase in runs, but we were sufficiently satisfied with r=150 for the purposes of this demonstration. Lastly, it seems Particle Swarm and Grid Search produce more precise estimates than the Genetic Algorithm on this particular problem. These GA results indicate we cannot confidently estimate the parameters (particularly Z) of the real data using the GA as it is currently specified. An analyst, seeing these results, should rerun these tests again under different hyper-parameter specifications δ or with r turned up further to see if estimate precision can be improved upon. The results of our exploration of alternative specifications of GA hyper-parameters on this problem in the Monte Carlo setting can be seen below.

Figure 3: MCS GA Results Using Alternate δs - Recency Bias (R)

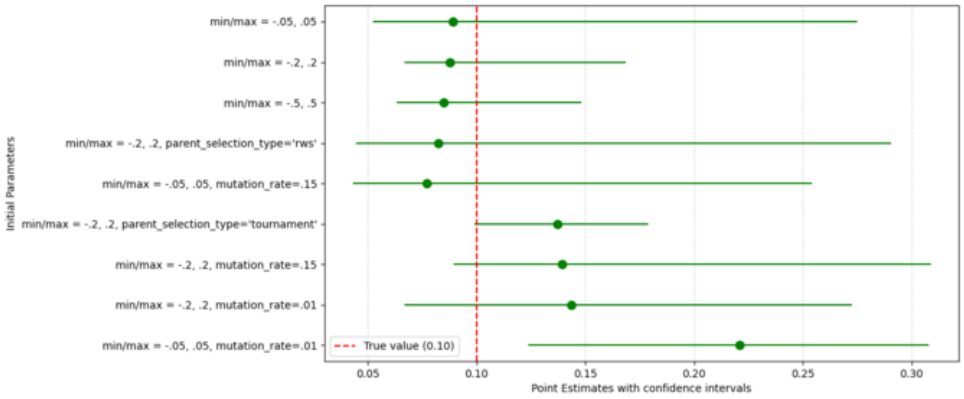
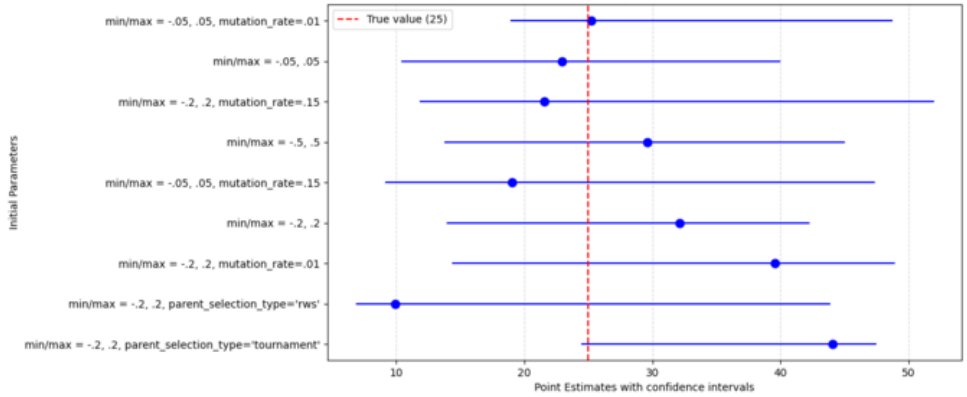


Figure 4: MCS GA Results Using Alternate δs - Prior Strength (Z)



None of these alternatives seems to improve on the precision with which Z is estimated in particular. While further exploration can always be done, this indicates perhaps a new search method should be tried for this problem (like one of the two alternatives we have explored above). This could also indicate that the model parameter Z cannot be reasonably identified. In our case, since we have shown these parameters can be recovered using other search algorithms, that is clearly not the case.

MCS Results for Estimate Bias: Next, we explore if our estimates are biased and if that bias is shrinking with run number. We provide two such plots below.

Figure 5: MCS Estimated Bias Results - Recency Bias (R)

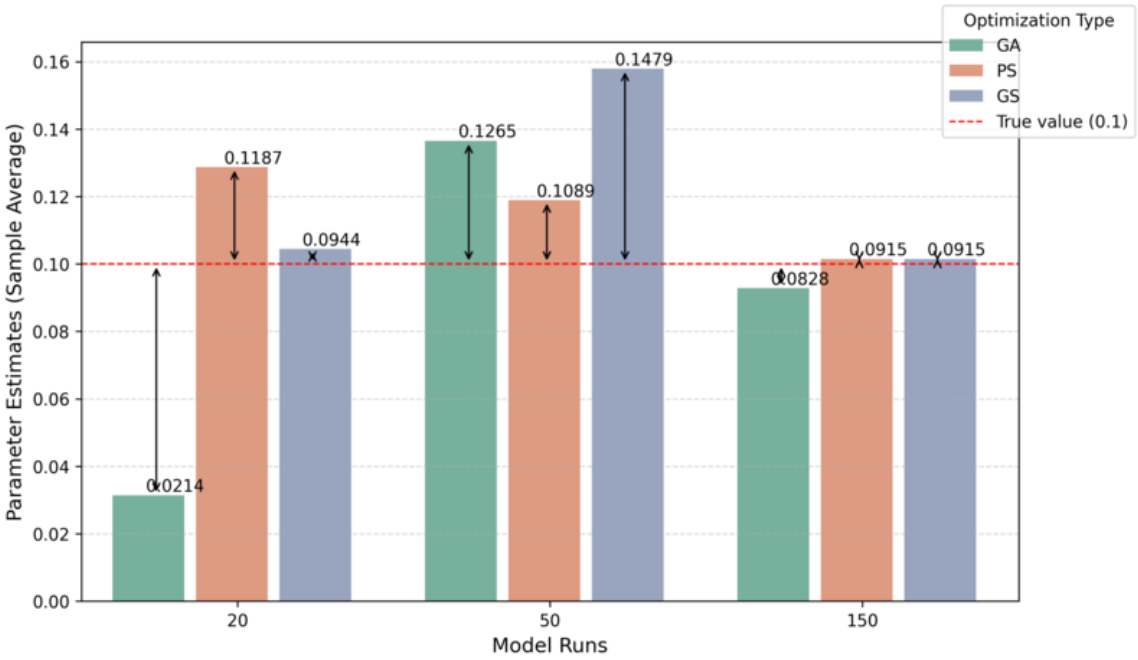
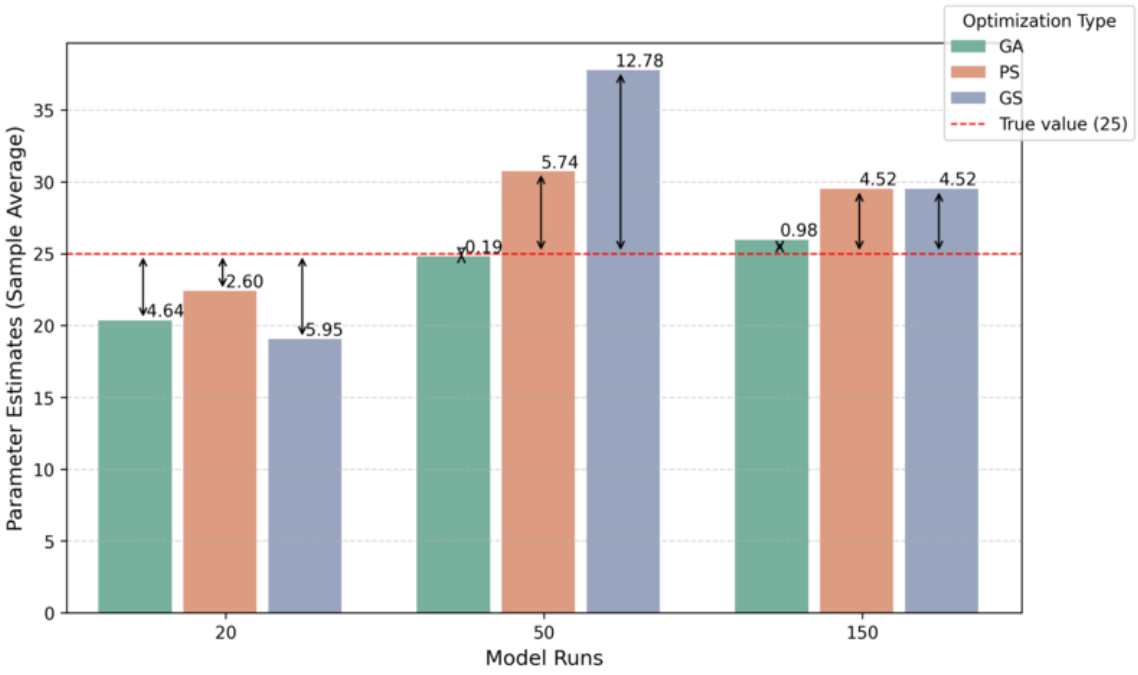


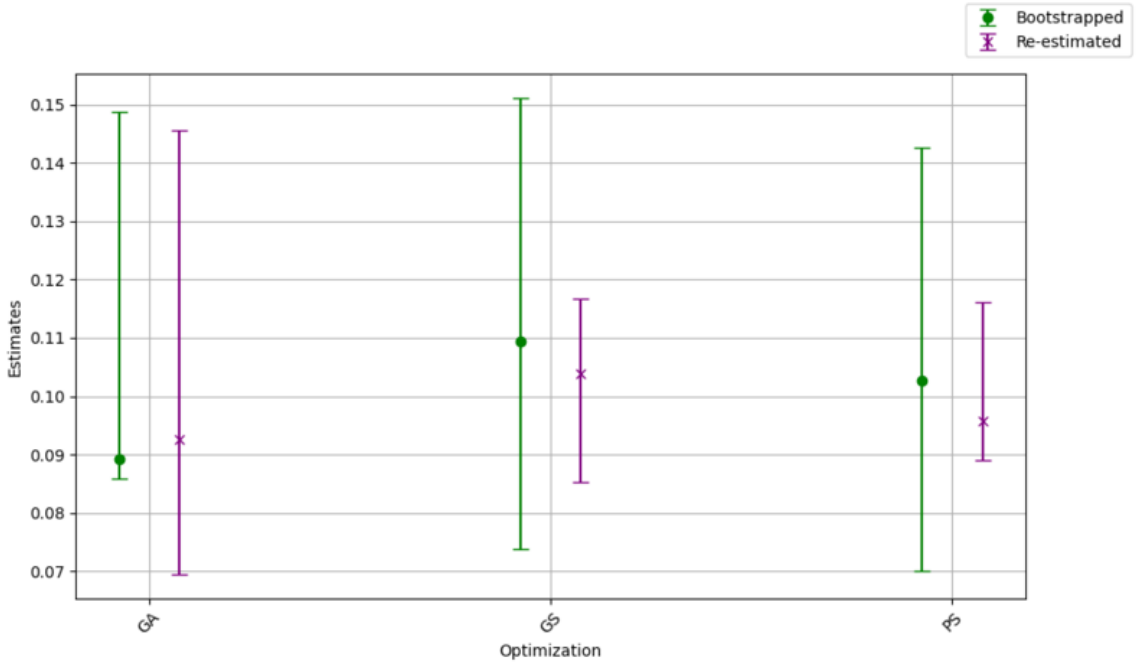
Figure 6: MCS Estimated Bias Results - Prior Strength (Z)



We see that while the bias in R seems to be decreasing in runs, the relationship between the bias in our estimates of Z and run number is not as clear. One possible way to remedy this is to simply subtract the recovered bias from our best estimates of Z that we get from data.

MCS Sources of Imprecision Test Results: We also demonstrate the results of our novel test for decomposing sources of estimate imprecision (Test 4) in the two plots below. Recall our goal is to identify how much of the imprecision in our estimates comes from variation in our data vs. variation in our model output and search process. While the width of the bootstrapped CIs contains all the above mentioned sources of variation, the CIs generated by simply re-estimating the same data many times should remove imprecision from sampling variation. The re-estimated CIs should tell us how much variation in estimate comes solely from our model and search process, while the difference in size when compared to the bootstrapped CIs should tell us how much imprecision can be attributed to sampling variation.

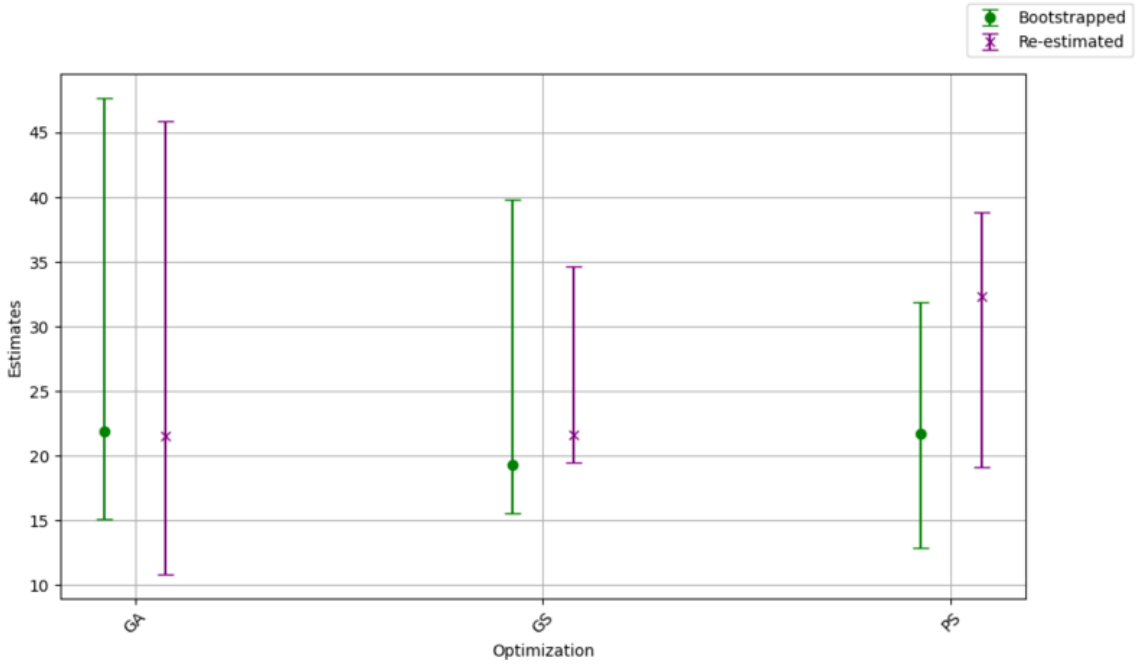
Figure 7: MCS Imprecision Source Comparison - Recency Bias (R)



Above, we see that for Gridsearch (GS) and Particle Swarm (PS), the role of sampling variation plays a substantial role in explaining the size of R's confidence intervals. For the GA however, it seems that the variation in the estimates of R it produces is very large even when applied on precisely the same data. This can be seen by looking at the relative size of the re-estimated CI. We also see that the bootstrapped CI for R using the GA, which has an additional source of variation, is not any larger[3].

[3]In fact it is smaller despite having an additional source of variation. This can be attributed to randomness involved in the construction of these CIs.

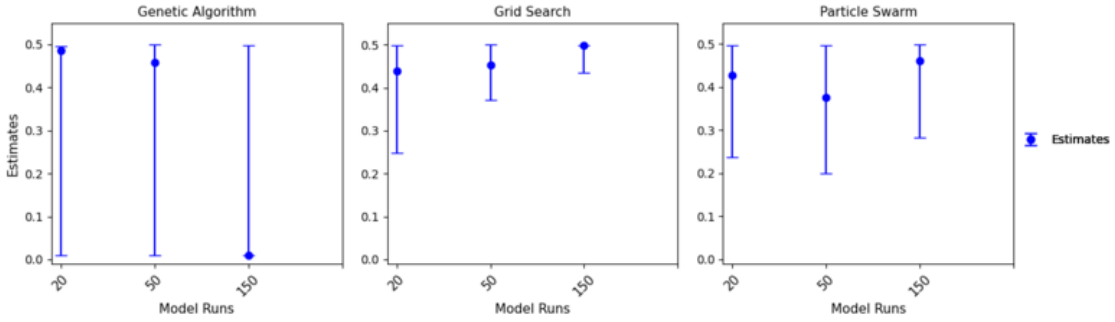
Figure 8: MCS Imprecision Source Comparison - Prior Strength (Z)



Once again, we see (above) the GA results span nearly the entire search space of Z, with nearly all of its variation attributable to how we search the parameter space with the GA. GS and PS, while featuring overall smaller CIs than the GA, demonstrate that for estimating Z, variation from model output and searching play a more substantial role in explaining the imprecision of the estimate.

Data Results: At this point, analysts using GS or PS as their $search(\cdot)$ process with these specifications could proceed with estimation of the real data with greater confidence. Again, the process of finding the best fitting parameters θ^* and CIs is precisely the same as was done in the Monte Carlo Simulations (Tests 1 and 2), though this time we use real world lab data. For completeness, we also report results for the GA, and once again do so across all combinations of runs and optimization techniques as was done above, which can be seen below.

Figure 9: Estimates on Data - Recency Bias (R)

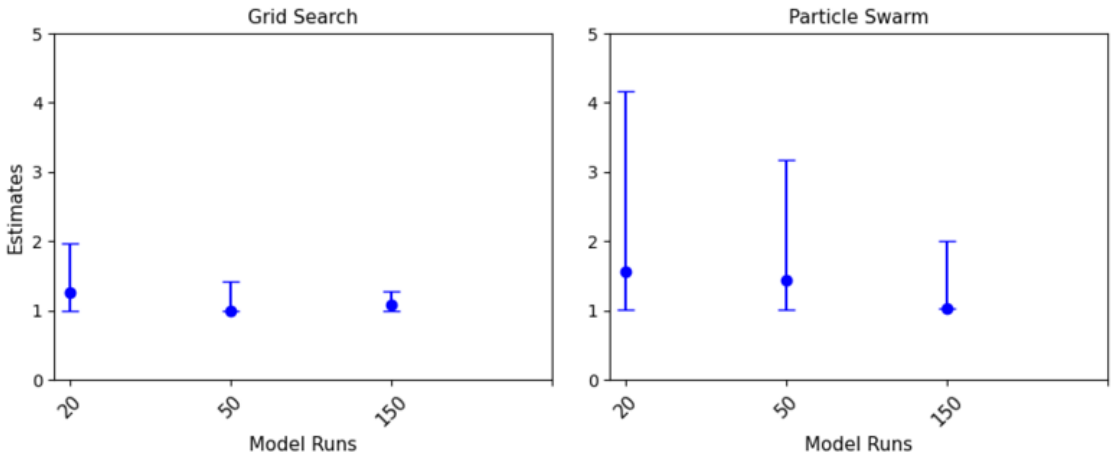


As indicated by our simulations, it seems Grid Search (GS) and Particle Swarm Optimization (PSO) at the highest level of runs (r=150) can produce much more precise estimates of R. Further, it seems both GS and PSO return similar estimates of R, with GS best estimating R = 0.50 with CI = $[0.43, 0.50]$ while PSO returns R = 0.46 with CI = $[0.28, 0.50]$.

Figure 10: Estimates on Data - Prior Strength (Z)

Once again, Grid Search (GS) and Particle Swarm Optimization (PSO) produce fairly precise estimates of Z at (r=150) while the Genetic Algorithm seems to produce CIs spanning nearly all possible values Z is allowed to take. Once again, it also appears that GS and PSO are in relative agreement, with GS best estimating Z = 1.04 with CI = $[1.02, 2.01]$ while PSO returns R = with CI = $[1.00, 1.27]$. Below we include the same plot 'zoomed in' to better see the results for GS and PSO.

Figure 11: Estimates on Data - Prior Strength (Z) Zoomed In



6 Application: Information Diffusion

6.1 Data and Background

As a second demonstration of our method, we bring a model of information diffusion (the 'cascade model') to data on the flow of information across a social network used in Rand et al. (2015).

The Data: We utilize one data set collected by and used in Rand et al. (2015) on Hurricane Sandy mentions on Twitter. Rand et al. (2015) uses a snowball network sample of 15,000 Twitter users to build the network. Next, using the Twitter API, current and past tweet information about Hurricane Sandy was identified by looking at keywords and then retained if it belonged to someone in the network, with a corresponding time stamp. Post-processing was then done to identify the first time an individual mentioned a keyword in a tweet or a retweet and how many people have mentioned the keyword over time to produce a time series of length 50. This is done on 100 different occasions, producing 100 time series of activation level in the network over varying lengths of time[4]. For the purposes of block-bootstrapping, all players in a given time series will serve as a *group* g with the number of groups $G = 100$, where each group is simply the individuals considered in each time series. Further, a *block* b_g is defined as all periods of play by players in one of the time series collected. These will be the units which we bootstrap to generate our confidence intervals.

[4]For further details on how these data were collected, see Rand et al. (2015).

The ABM: From the data above, we have a network sampled from its real-world parallel. We explore the cascade model (first proposed in Goldenberg et al. (2001)) which aims to capture the dynamics of information diffusion over a network in two parameters.

In this model, each agent (node) either knows about Hurricane Sandy or does not during any given period t, meaning their state $n_{i,t} \in \{0, 1\}$. The model is initialized with all deactivated agents, meaning $n_{i,t=0} = 0 \; \forall \; i$. Next, a number of agent(s) are randomly selected to activate to correspond with the initial level of activation in the time series. Once an agent is activated, they remain activated for the duration of the model.

Each round, agents who are not activated have one of two ways to possibly learn about Sandy and become activated: agents can learn from an *outside source* or *from their neighbors*.

At the start of each period t, agents have a chance **p** of learning from a source outside of the network each round, which Rand et al. (2015) refers to as *innovation*.

Next, each agent that is still not activated has some probability **q** to learn from their neighbors if at least one of their neighbors was activated in the previous time step. Neighboring agents which were activated two or more periods ago do not serve as sources of activation. This type of activation is referred to as *imitation*.

Unlike in Rand et al. (2015), we assume that p q are constant in the population. The probability that any given agent i is activated after these steps can be formalized as follows:

$$Prob_{i,t}(n_{i,t} = 1) \begin{cases} 1 & \text{if } n_{i,t-1} = 1 \\ 1 - (1-p)(1-q) & \text{if } n_{i,t-1} = 0 \text{ and if } \exists \; n_j \in N(n_i) \text{ s.t. } n_{j,t-1} = 1 \text{ and } n_{j,t-2} = 0 \\ p & \text{otherwise} \end{cases} \tag{20}$$

where $N(n_i)$ refers to the neighboring nodes of n_i.

As seen above, our parameters p (innovation) and q (imitation) will make up the vector of parameters θ that we want to estimate on our data, taking this structural model of information diffusion as given.

Note, unlike in Rand et al. (2015), we assume p and q are constant across the cascades collected for our example problem. This is done so we can leverage these data to generate confidence intervals. As we will see, this assumption may be problematic.

The Structural Assumption: This second exercise has more structural assumptions underlying the empirical exercise, as one might expect when moving from a controlled lab environment to data gathered from the real world. As before, we are assuming that the behavior that the cascade model can capture, governed by the same p and q values, can reasonably approximate the true DGP_D that generated these data (the actual process that governed the tweet behavior). Additionally, we must recognize that this network is a sample of a

much larger social network both online and offline. Given this, we must also assume that the sampled network is representative of the true underlying network. This includes assuming that p, the chance of learning from an outside source, need not correspond to some offline network and that all agents are equally likely to learn from outside sources, or at least that this difference doesn't matter so much. Finally, when bootstrapping over blocks which have overlapping members in their groups (in this case, completely overlapping), we also assume that behavior in each of these 100 cascades is reasonably independent. Since these groups for each block contain precisely the same set of individuals between blocks, this means we are implicitly assuming there is not any substantial learning or adaptation in behavior by members of these groups over the period which these samples were collected.

On Estimating Best Fitting Parameters: During this exercise, we examine three different summary functions that vary only in the number of moments of the data they capture for incorporation into fitness. Specifically, our *summary function* $S(\cdot)$ uses the first one, two, or three moments of the activation level. The output of these functions is a vector (or set of vectors) of length T, where each entry represents the corresponding moment, mean, variance, or skewness, of the activation level at a given timestep.

We then apply our *aggregation function* $Agg(\cdot)$ which computes the average of each of these moments in time across a fixed number of model runs (r = 50).

Next, we explore three different versions of our original *fitness function* $fit(\cdot)$, incorporating either the first one, two, or three moments into the fitness calculation. Mechanically, this operation is done by computing the mean squared error between the vectors of moments of activation rate over time created by summarizing our data and summarizing aggregated model runs, where each moment's weight in the fitness function is normalized. As described above, this can be formalized as follows:

$$fit(\theta) = \frac{1}{T+1}\sum_{t=0}^{T}[\alpha_{1,t}(\bar{\mu}_{M_A,1,t} - \bar{\mu}_{D,1,t})^2 + ... + \alpha_{n,t}(\bar{\mu}_{M_A,n,t} - \bar{\mu}_{D,n,t})^2] \quad (4)$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) = \{\bar{\mu}_{M_A,1,t=0}, \bar{\mu}_{M_A,2,t=T}, ...\bar{\mu}_{M_A,n,t=0}, ..., \bar{\mu}_{M_A,n,t=T}\} \quad (5)$$

and where,

$$\alpha_{j,t} = \frac{1}{\bar{\mu}_{M_A,j,t}} \quad (6)$$

Finally, we once again explore the effect of choice of optimization technique, evaluating performance across the three $search(\cdot)$ specifications considered in our first example (grid-search, the genetic algorithm, and particle swarm optimization).

On Bootstrapping: Following the steps laid out above in Section 3.2, bootstraps can be performed by constructing a resampled dataset using our 100

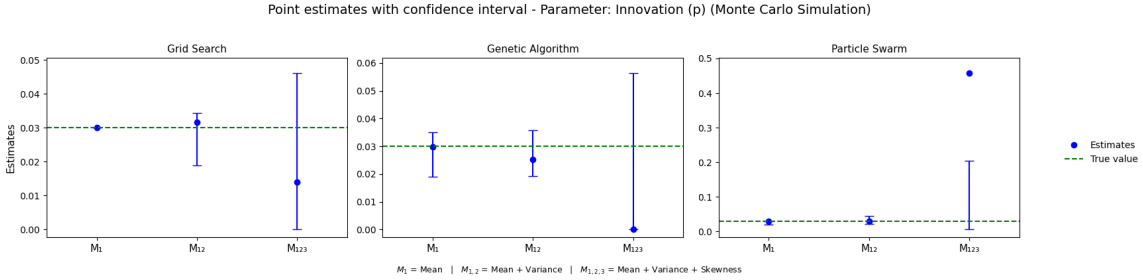
blocks, then searching for parameters which best fit the data. As before, we choose to re-sample 100 times (that is, k = 100). Then, for each parameter, we drop the outside 5% of estimates, and the remaining extremes serve as the 95% confidence intervals.

On Monte-Carlo Simulation: To investigate estimator performance, we run our Monte Carlo simulation as discussed in section 4, using $p =$ and $q =$. As a reminder, this is done by first producing a 'simulated data set', using resamples of that data as if it were real data to get estimates, and then using those estimates to generate confidence intervals. At the end, we have a signal of how well the known values for p and q can be recovered.

6.2 Results

MCS Results for Best Fits and CIs: We begin with the results for both our best estimates (Test 1) and our confidence intervals (Test 2), though this time we explore the interaction of three optimization techniques (GA, GS, and PS) with three fitness function specifications.

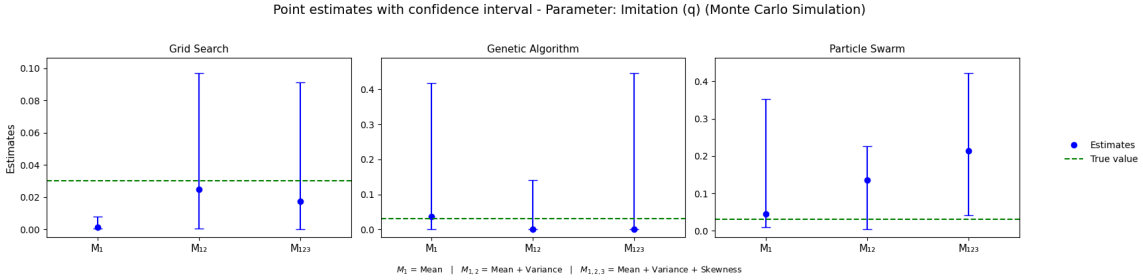
Figure 12: Monte Carlo Simulation Results - Innovation (p)



For p, we can see above that all search algorithms produce fairly accurate point estimates (estimates close to the true value $p = 0.03$) when fitness only incorporates the first moment of the data. We also notice, however, that the confidence intervals for GS appear to be near width zero and exclude the true value of p. When we include the second moment of the data in the fitness function, we see the point estimates for p remain fairly accurate, and the confidence intervals are both fairly precise and contain the true value of p across all *search* techniques. Finally, when the first three moments are incorporated into the fitness function, we observe less accurate point-estimates of p across the board with PS performing fairly poor. Overall, it appears many of these specifications produce reasonable estimates for p.

Looking at performance for q, we see a distinctly different picture.

Figure 13: Monte Carlo Simulation Results - Imitation (q)



First, recall that the true value of $q = 0.03$ was used in these MCSs. While point estimates for q when only using the first moment in fitness appear to be somewhat reasonable, we can see that their some of the associated CIs are not. GA and PS both produce fairly large confidence intervals, indicating a fairly poor level of estimate precision. The CIs from GS may appear to perform better as they are very 'precise', but they do not include the true, known value $q = 0.03$. None of these specifications seem suitable for recovering q. Moving to a fitness function which incorporates the first two moments of data, however, seems to generate more reasonably precise CIs across *search* specifications, all of which contain the true value. In PS and GA, however, we see less precise point-estimates for q. When the third moment is added into the fitness function, we see this pattern continue, with the CIs associated with GA a PS spanning nearly the entire parameter space without any gains to the accuracy of the point estimates. GS sees only a small change in performance, with a point estimate further from 0.03 and a similar confidence intervals to before.

Taken together with the results for p, it is probably best to proceed with using the first two moments in the fitness function, as using only the first or the first three can lead to fairly poor levels of estimate precision of q and to a lesser degree p. Comparing *search* algorithms, perhaps a slight preference can be given to GS over GA as our tests indicate it performed a bit better at recovering q.

To demonstrate the importance of fitness function specification on the identifiability of model parameters, we show a number of alternative specifications which were tried using only the first moment in the fitness. Across all specifications, imprecise estimates of q remained a persistent problem. This illustrates the importance of getting the fitness function 'right.'

Figure 14: MCS GS Results Alternative Runs - Innovation (p)

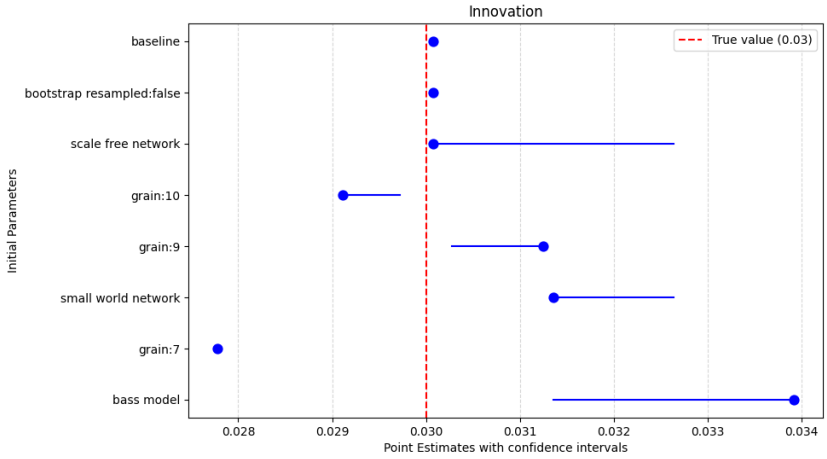
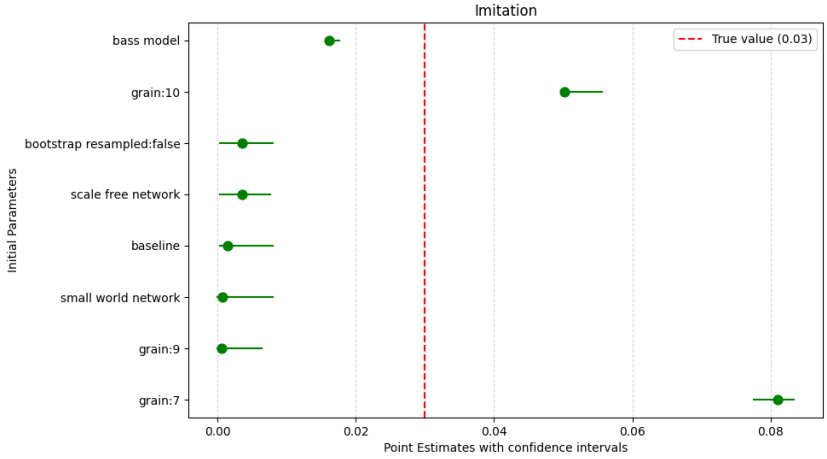
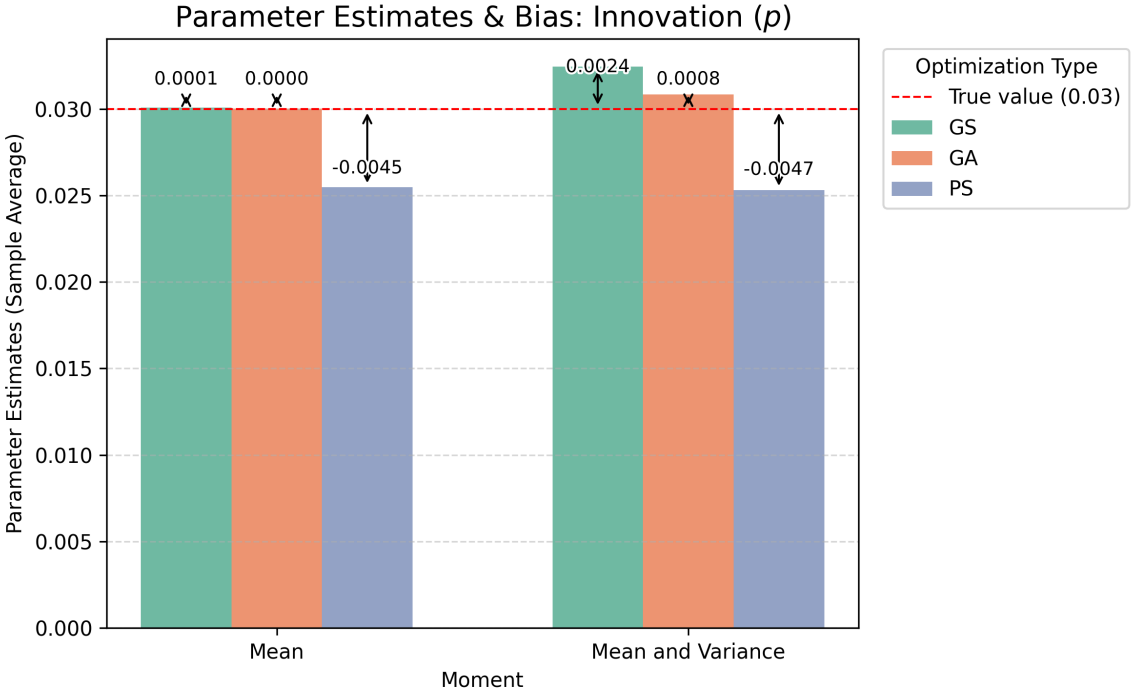


Figure 15: MCS GS Results Alternative Runs - Imitation (q)



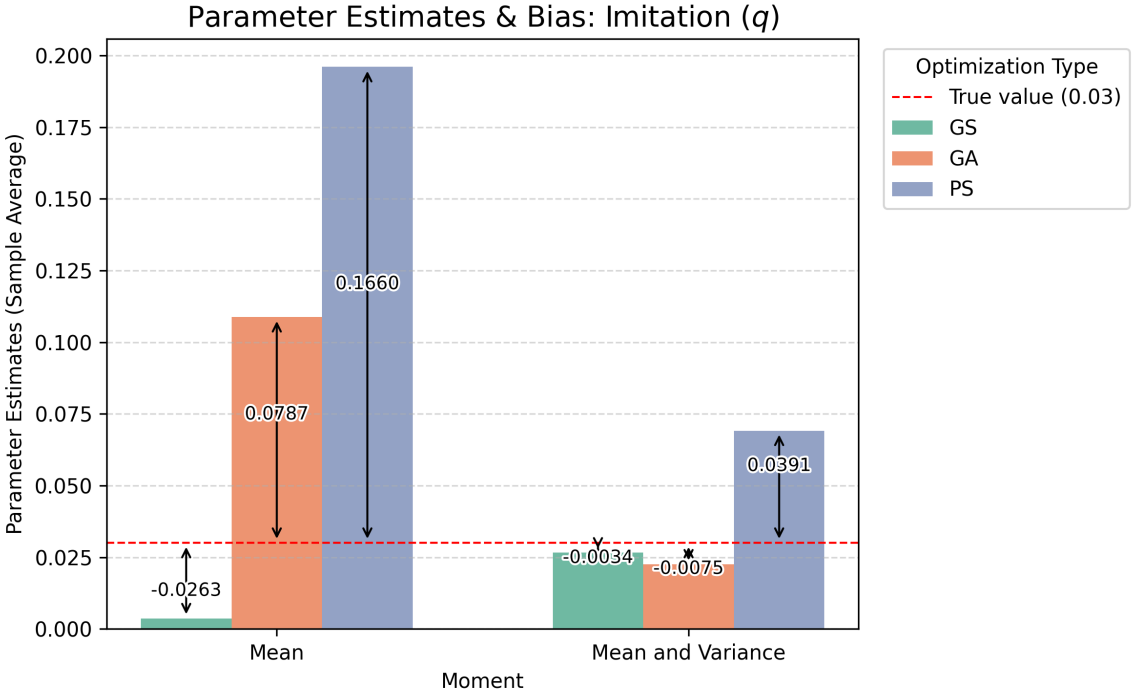
MCS Results for Estimate Bias: Next, we take a look at our bias estimates.

Figure 16: MCS Estimated Bias Results - Innovation (p)



First, we note that across run levels, our estimates of p are fairly unbiased, with only PS reporting modest levels of underestimation. Next, let us look at q.

Figure 17: MCS Estimated Bias Results - Imitation (q)

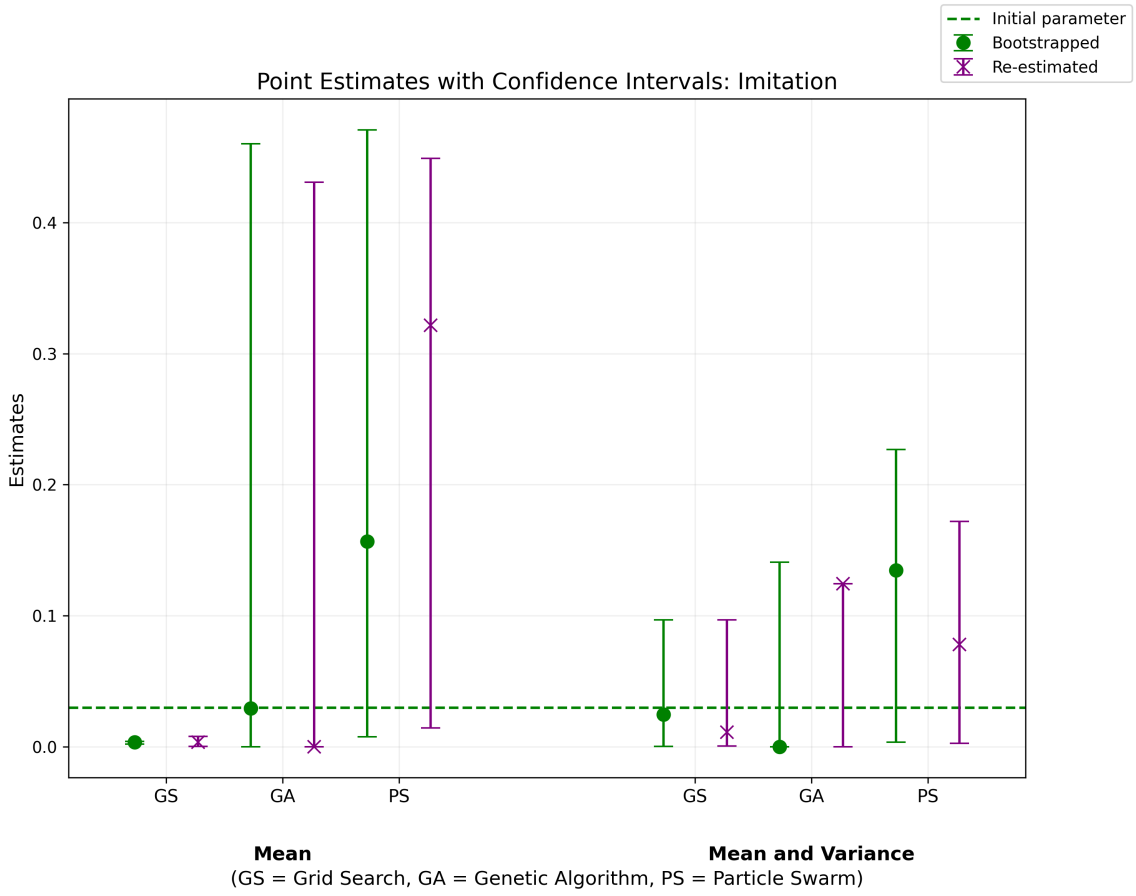


We can see a fairly large degree of variation in bias in q estimates across our *search*() choices when only considering the mean in the fitness function. We can see the degree of bias shrinks quite a bit, however, when also including the second moment in the fitness function. GS and GA appear particularly capable of producing reasonably unbiased estimates of q when the second moment is also considered in fitness.

MCS Sources of Imprecision Test Results: Next, we can learn something about the sources of estimate imprecision (Test 4). Recall our goal is to identify how much of the imprecision in our estimates comes from variation in our data vs. variation in our model output and search process. We start with q first this time, whose output is more aligned with what we might expect.

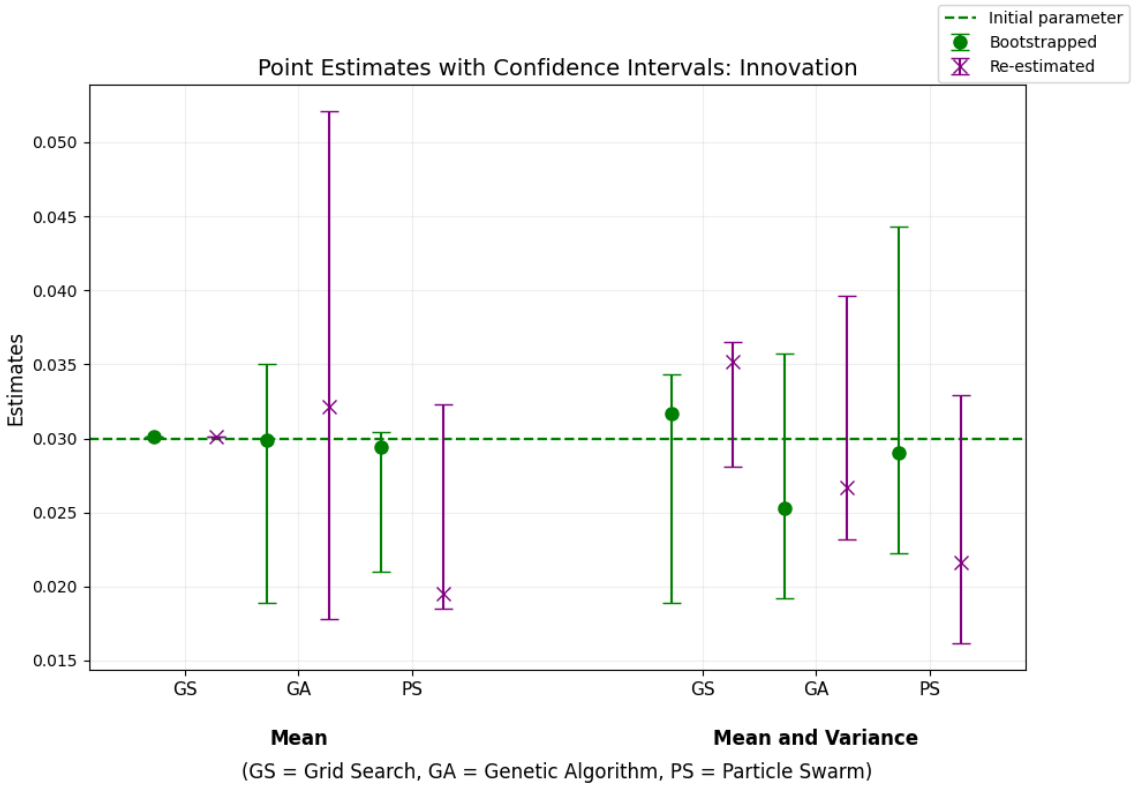
We see first, as we did above, there's an increase in precision for GA and PS when we add the second moment which continues to be true for our re-estimated results. We also see GS's CIs include the true value as we move to two moments. Comparing bootstrapped CIs to their re-estimated equivalents, we see similar or very small gains in precision for GA, PS, and GS (in the two moment case), with the only non-trivial difference occurring for PS when using Mean and Variance. This seems to indicate that a great deal of the variation in estimates comes from noise in the model and search process rather than from variation in the data.

Figure 18: MCS Imprecision Source Comparison - Imitation (q)



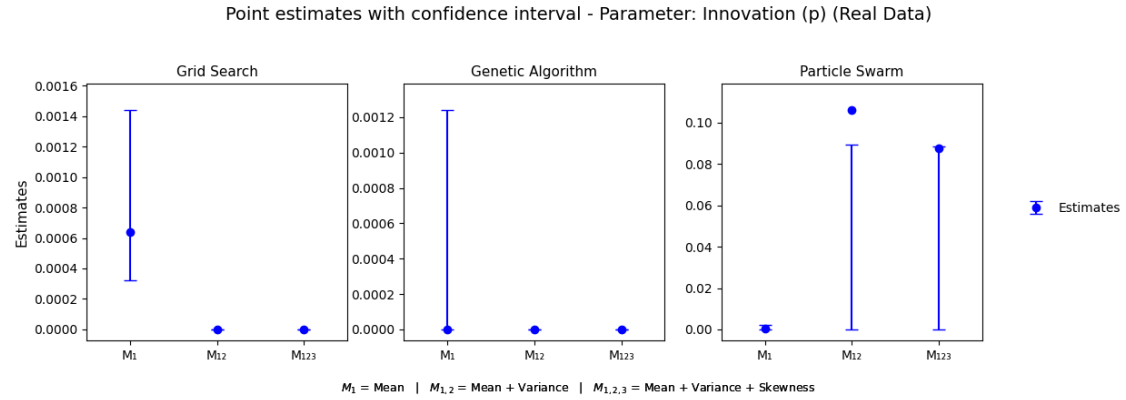
Next taking a look at p. Across the board, we can see precision is much better when both mean and variance are considered for both GA and PS, while the inclusion of variance leads to the inclusion of the true value for GS. We can also see that much of the imprecision in the GA and GS when using mean and variance comes from model and search process noise itself, as evidenced by the similar sizes of these intervals.

Figure 19: MCS Imprecision Source Comparison - Innovation (p)



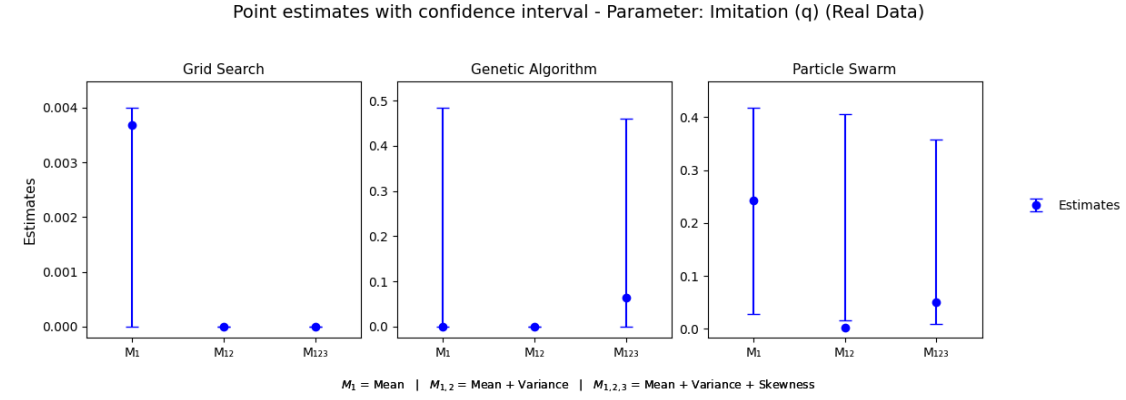
Data Results: Taking our lessons from the MCSs we ran above, let us try to bring our model to the real data. We see fairly precise point estimates across the board for GS and GA, with our favored mean and variance versions indicating the best-fitting value of p is 0. We also may notice some strange inversion of the CIs we saw in our MCSs. When we did our MCSs, CIs seemed to widen when moving from one to two moments. When brought to data, however, the opposite occurs. This doesn't seem consistent with what should happen if this model were a good approximation of the underlying DGP. Let us take a look next at q.

Figure 20: Estimates on Data - Innovation (p)



Once again, we see fairly precise estimates when using GS and GA, with our favored mean and variance version of our fitness function indicating that the best fitting values for q is 0. And once again we see this strange inversion of effect in CIs. CIs in the MCS got wider moving from one moment to two in our fitness function, but here for GS and GA, the opposite occurs.

Figure 21: Estimates on Data - Imitation (q)



So what does this all tell us? The MCS tells us what would happen if the data were generated if our structural model were identical to the true DGP, and we saw that we *should* be able to recover the true parameters of p and q using certain combinations of search algorithm and fitness function specification (in particular, GS or GA with two moments in the fitness function). When we actually bring the model to data using these specifications, however, not only do we see estimates of 0 for p and q, but the overall behavior of the estimation techniques is inconsistent with how it should be

performing given the model is a good approximation of the real underlying DGP. This leads us to conclude that perhaps this model, with p and q assumed to be constant across cascades, is simply not valid in this context.

7 Outlook and Conclusion

This paper underscores that as computational models, and especially agent-based models, continue to move from proofs-of-principle toward direct empirical estimation, careful attention must be paid to their estimation properties. Throughout this work, we have demonstrated that Monte Carlo simulation is an indispensable tool for analysts seeking to establish parameter identifiability, evaluate algorithm performance, and assess the precision and accuracy of their estimates across different contexts. We have also shown in our second example a case where an identifiable model performed fairly differently when brought to data, indicating the proposed model may not be a valid representation of the underlying DGP. Our findings also highlight that it is not always obvious whether an ABM is empirically identifiable, and that even well-identified models can produce vastly different results when estimated with different estimation techniques. This underscores the danger of relying on standard methods without verifying their suitability for a given application.

More broadly, this work advances Monte Carlo testing as a standard diagnostic for assessing the properties of computational model estimators. We hope that the tools and practices outlined here will help bridge the gap between the conceptual power of ABMs and their rigorous application to data, ultimately strengthening the role of computational modeling across the social sciences.

References

Gabriele Camera and Marco Casari. Cooperation among strangers under the shadow of the future. *American Economic Review*, 99(3):979–1005, June 2009. doi: 10.1257/aer.99.3.979. URL https://www.aeaweb.org/articles?id=10.1257/aer.99.3.979.

Ido Erev and Alvin E. Roth. Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *The American Economic Review*, 88(4):848–881, 1998.

Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001. ISSN 1573-059X. doi: 10.1023/A:1011122126881. URL https://doi.org/10.1023/A:1011122126881.

Christian Gourieroux and Alain Monfort. *Simulation-Based Econometric Methods*. Oxford University Press, 1996.

Daniel McFadden. A method of simulated moments for estimation of discrete response models without numerical integration. *Econometrica*, 57(5):995–1026, 1989. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1913621.

John H. Miller and Scott E. Page. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton University Press, 2007. ISBN 9780691127026.

William Rand, Jeffrey Herrmann, Brandon Schein, and Neža Vodopivec. An agent-based model of urgent diffusion in social media. *Journal of Artificial Societies and Social Simulation*, 18(2):1, 2015. ISSN 1460-7425. doi: 10.18564/jasss.2616. URL http://jasss.soc.surrey.ac.uk/18/2/1.html.

Iza Romanowska, Colin D Wren, and Stefani Crabtree. *Agent-Based Modeling for Archaeology: Simulating the Complexity of Societies*. The Santa Fe Institute Press, August 2021. ISBN 1947864254. doi: 10.37911/9781947864382.

H. Sayama. *Introduction to the Modeling and Analysis of Complex Systems*. Open SUNY Textbooks. Open Suny Textbooks, 2015. ISBN 9781942341093. URL https://books.google.com/books?id=Bf9gAQAACAAJ.

Thomas C. Schelling. Dynamic models of segregation†. *The Journal of Mathematical Sociology*, 1(2):143–186, 1971. doi: 10.1080/0022250X.1971.9989794.

Karl Schmedders and Kenneth L. Judd. *Handbook of Computational Economics, Volume 3*. North-Holland Publishing Co., NLD, 2014. ISBN 978-0-444-52980-0.

Leigh Tesfatsion and Kenneth L. Judd. *Handbook of Computational Economics, Volume 2: Agent-Based Computational Economics (Handbook of Computational Economics)*. North-Holland Publishing Co., NLD, 2006. ISBN 0444512535.

Uri Wilensky and William Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. The MIT Press, 2015. ISBN 9780262731898. URL http://www.jstor.org/stable/j.ctt17kk851.

Christopher Zosh, Nency Dhameja, Yixin Ren, and Andreas Pape. Agentcarlo: A python package for estimating parameters and confidence intervals for agent-based models (working paper). 2025.