JASSS
THE JOURNAL OF ARTIFICIAL SOCIETIES AND SOCIAL SIMULATION

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

Christopher Zosh, Nency Dhameja, Yixin Ren, Andreas Pape[a 1]

[1] *Binghamton University, United States*
Correspondence should be addressed to *czosh1@binghamton.edu*

Journal of Artificial Societies and Social Simulation xx(x) x, 20xx
Doi: 10.18564/jasss.xxxx Url: http://jasss.soc.surrey.ac.uk/xx/x/x.html

Received: dd-mmm-yyyy Accepted: dd-mmm-yyyy Published: dd-mmm-yyyy

[a] 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.

Abstract: Agent-based models (ABMs) are increasingly used for formal estimation and inference, but their complexity and algorithmic nature pose persistent challenges for the formal assessment of estimator properties.

This paper highlights the indispensable role that Monte Carlo simulations (MCS) can play in addressing these challenges. We show that MCS can systematically evaluate whether parameters of an ABM can be reliably estimated, as well as how estimate accuracy and precision depend on factors such as search algorithm choice and the number of model runs conducted. We also introduce a novel Monte Carlo test that disentangles imprecision due to the stochasticity of the model and estimation process itself versus that sourced by sampling variation.

We apply these techniques to two example applications: first, a repeated prisoner's dilemma model with learning agents and second, a model of information diffusion over a network. Our results demonstrate that, while the parameters of these models can be identified in principle, estimator performance can be highly sensitive to choice of hyper-parameters used in the estimation process and to features of the model itself. These findings underscore the practical importance of applying MCS-based diagnostics before drawing substantive conclusions from estimated ABM parameters.

Keywords: Agent-Based Modeling, Estimation, Bootstrapping, Monte Carlo Simulation, Simulated Method of Moments, Monte Carlo simulation, identification

Introduction

1.1 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, his model was performed using coins on a chess board. 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 in economics.

1.2 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. Although notable progress has been made in developing agent-based econometric methods (e.g. (Bargigli 2017)), there is still a substantial need for accessible and computationally feasible best practices for using ABMs in this way. One key difficulty is that ABMs often cannot be fully described as a set of equations, which complicates the formal assessment of parameter identifiability and estimator properties.

1.3 The primary goal of this paper is to highlight the indispensable role of Monte Carlo simulation (MCS) 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 and number of model runs. Such analysis is vital for determining the extent to which recovered parameter estimates reflect their real-world counterparts, which we demonstrate in our example applications described below.

1.4 Our findings underscore that optimization algorithm 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.

1.5 The remainder of this paper is structured as follows. Section discusses the existing literature and outlines our contribution. Section provides a brief overview of parameter estimation techniques for computational models and their associated confidence intervals. In Section , 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.

1.1 Sections and apply these MCS-based tests to two different ABMs. In the first application, we detail an ABM of learning agents playing a repeated prisoner's dilemma in small groups, drawing on data and design from Camera and Casari (2009). Our results show that although there is evidence that the model is structurally identifiable, only some algorithm configurations produce estimates that are both reasonably accurate and precise. This underscores the importance of computationally verifying identifiability rather than assuming it. In the second application, we explore a model of information diffusion across a network. Here we again find that the parameters can be recovered, but that once again, estimation performance depends on the chosen algorithm and that relative performance differs from the first application. This sensitivity showcases the importance of this tool as a way to demonstrate the effectiveness of any particular estimation process (including search algorithm specification) in the context of its intended application rather than taking on faith that it will perform reasonably. Together, these results highlight the practical need to conduct such tests across different contexts before drawing substantive conclusions.

Literature Review

2.1 How best to bring ABMs to data has remained in many ways illusive and fairly non-standardized. 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 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.

2.2 In the economics 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, 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]

2.3 Because of this, many analysts utilizing ABMs see Simulated Method of Moments (SMM), a fairly generalizable method for estimating parameters, as their go-to estimation technique of choice. SMM was first introduced in

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

McFadden (1989) and is summarized in a number of econometric texts, including a classic text on simulation-based methods (Gourieroux and Monfort 1996). Although these texts provide thorough coverage of some aspects of SMM, they may prove challenging for non-econometricians to engage with and are not adapted to address the challenges that arise when trying to estimate ABMs in particular. Therefore, we feel part of our value added is to lay bare this method for parameter estimation and to discuss its application and interpretation in the context of ABMs. This includes providing a number of useful insights, particularly on how to interpret the elements and outputs of ABM estimation, many of which are summaries or extensions of a number of ideas from the existing structural estimation literature within and outside economics (including Hoyle (2012) and Greene (2017)).

2.4 We also formalize the application of block-bootstrapping for critical values in the estimated parameters and related Monte-Carlo simulations, which we argue are tools ideally suited for such contexts. When generating these discussions, a great deal of attention was given in particular to Davidson and MacKinnon (2002) and MacKinnon (2006). When discussing Monte-Carlo simulation, we make no claim that we are the first to use such methods in the context of ABMs. Many sources, including those mentioned above, make reference to utilizing such methods in one way or another. To the best of our knowledge, however, there are no existing tests for decomposing estimate imprecision sources for simulation models as we have proposed (Test 4) in Section 4.23.

ABM Estimation Overview

3.1 The goal of this section is to give a brief 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]

Estimating Best-Fitting Parameters

3.2 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* $S(.)$, an *aggregation function* $Agg(.)$, a *fitness function* $fit(.)$, and an *optimization technique* $search(.)$.

3.3 A *summary function* $S(.)$ 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 as is common practice. We denote these summarized outputs from the model and data $Z(Y_{M_A}, \theta)$ and $Z(Y_D)$ respectively.

3.4 An *aggregation function* $Agg(.)$ 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 $\bar{Z}(Y_{M_A}, \theta, r)$.

3.5 A *fitness function* $fit(.)$ 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 and for the examples included in this paper, fit only the first moment of output for each time step of model and data. This is given by:

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

[2] For a more detailed discussion of much of this section, please see ...

where,

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

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

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

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_D)), \delta) \rightarrow \theta^* \quad (4)$$

where

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) \quad (5)$$

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.

Bootstrapping Confidence Intervals

3.8 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 established. 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 estimate 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.

Establishing Properties with Monte Carlo Simulations

4.1 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 noise likely enters non-trivially, and an estimation technique $search(\delta, .)$ 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.

4.2 Instead, we can run a number of tests which utilize Monte Carlo Simulation (MCS), which can be used 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 increase our runs r or under other search parameters.

Monte-Carlo Simulation - What is it?

4.3 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.

4.4 In our case, we know that $search(\delta, .)$ may return different results of θ^* (see equation 4), even when fitting the same data, due to the stochastic nature of both $search(.)$ and our model M_A. Thus, we can think of $search(.)$, 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.

4.5 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.

4.6 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.

4.7

$$M_A(\theta, X) \rightarrow \hat{D} \tag{6}$$

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.

Test 1 - Evaluating Estimate Accuracy

4.8 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.

4.9 To do this, we simply apply our estimation technique $search(.)$ 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,

4.10

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\hat{D}})), \delta) \rightarrow \theta^* \tag{4}$$

4.11 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 $agg(.)$ or trying new search parameters δ or search methods $search(.)$) may be required to achieve better results. If the issue persists, then you may have model parameters which are not reasonably identifiable. We demonstrate what results for this test on an example application for a number search algorithms and number of runs in section 6.9.

Test 2 - Evaluating Estimate Precision

4.12 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.

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

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

4.14 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:

$$\tilde{\hat{D}}_k = \{b^1, ...b^G | b^m \overset{\text{iid}}{\sim} U(B)\} \tag{7}$$

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

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

4.16

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\tilde{\hat{D}}_k})), \delta) \rightarrow \theta_k^* \tag{8}$$

3. Construct the Critical Value(s)

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

$$C_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{9}$$

4.18 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 $search(.)$ 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 the a novel test to decompose this imprecision which is presented later (Test 4).

4.19 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 $search(.)$ and δ.

Test 3 - Evaluating Estimate Bias

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

4.21

$$Bias^i = E(\theta^{i*}) - \theta^i \tag{10}$$

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

4.23 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*}.

$$Bias_m^i = \frac{1}{N}\sum_{j=1}^{N}\theta_j^{i*} - \theta^i \tag{11}$$

Test 4 - Decomposing Sources of Estimate Imprecision

4.24 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 re-sampling) 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 $search(.)$ 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 are slightly different 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.

4.25 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 $search(.)$ play a role in estimate imprecision.

4.26 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:

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

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

4.28
$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\hat{D}_k})), \delta) \rightarrow \theta_k^* \tag{12}$$

2. Construct the Critical Value(s)

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

$$\hat{C}_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{13}$$

4.30 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.

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

Example Application 1 - Repeated Prisoner's Dilemma Model with Learning Agents

Application Set Up

5.1 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).

5.2 **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)

5.3 **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. 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:

5.4 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:

Table 1: Payoff matrix for the Prisoner's Dilemma

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

5.5

$$Score_{i,t=0}(\hat{a}) = Z \tag{14}$$

5.6 where Z is some constant.

5.7 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:

5.8

$$Prob_{i,t}(\hat{a}) = \frac{Score_{i,t}(\hat{a})}{\sum_{\forall a} Score_{i,t}(a)} \tag{15}$$

5.9 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:

5.10

$$Score_{i,t+1}(\hat{a}) = \begin{cases} (1-R) * Score_{i,t}(\hat{a}) + \pi_{i,t} & \text{if } \hat{a} = a_{i,t} \\ (1-R) * Score_{i,t}(\hat{a}) & \text{otherwise} \end{cases} \tag{16}$$

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

5.12 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.

5.13 **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.

5.14 **On Estimating Best Fitting Parameters:** If the goal is to understand behavior, then the chosen *summary function* $S(.)$ 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.

5.15 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* $Agg(.)$ to the summarized results from these model runs which simply averages the cooperation rate across runs for each period of play.

5.16 Next, define our *fitness function* $fit(.)$ which takes the (the summarized data and the summarized, aggregated model output) and computes mean squared error between them.

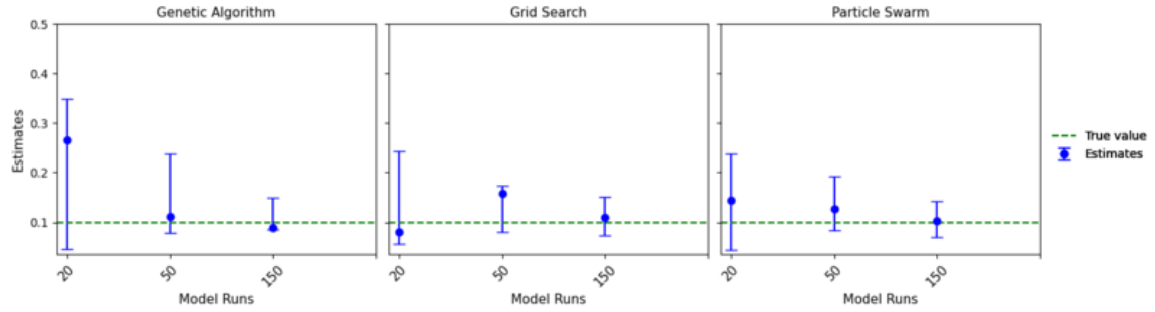
5.17 Last, an *optimization technique* $search(.)$ must be chosen to search for our best fitting parameters. We report results for three such optimization techniques: grid-search, the genetic algorithm, and particle swarm optimization.

5.18 **On Bootstrapping:** Following the steps laid out above in Section **??**, 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.

5.19 **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 is discussed in section section. 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.

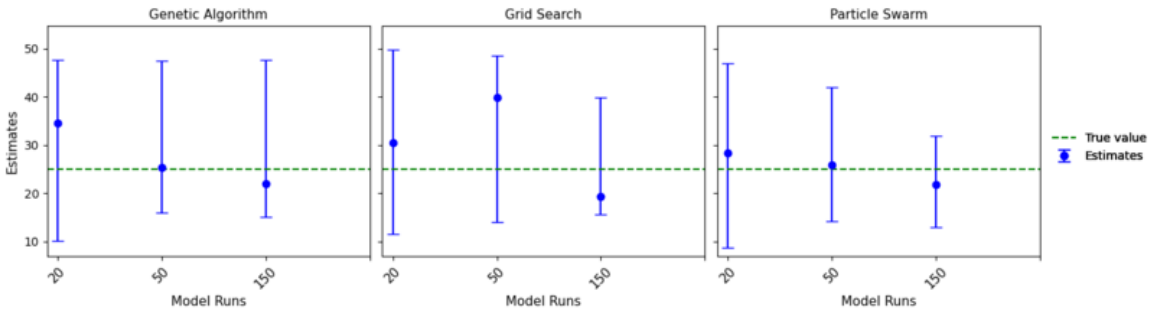
5.1 **MCS Results for Best Fits and CIs:** First, we show the results for both our best estimates (Test 1) and our confidence intervals (Test 2) from our Monte Carlo Simulations. We explore three levels of runs used in aggregation and three different optimization techniques, each with their own search 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)



5.2 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)



5.3 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 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)

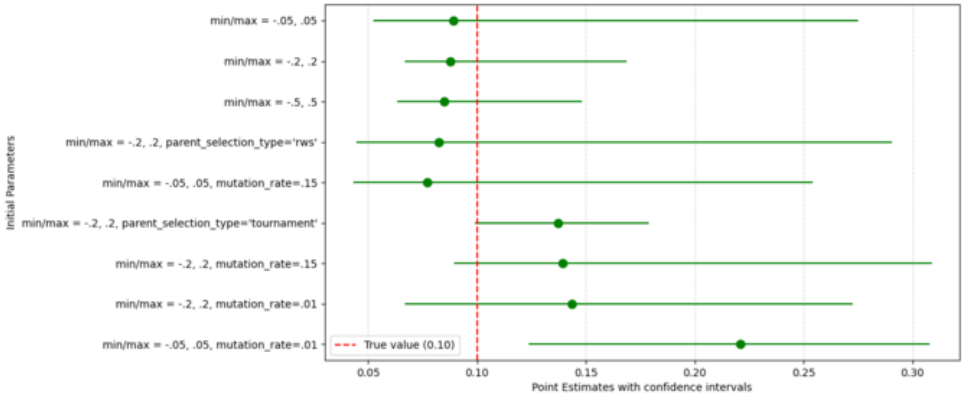
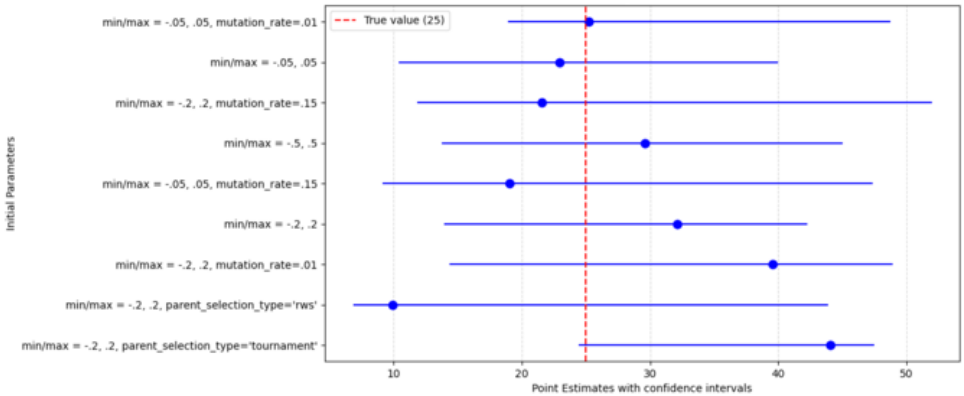


Figure 4: MCS GA Results Using Alternate δs - Prior Strength (Z)



5.4 None of these alternatives seem 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.

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

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

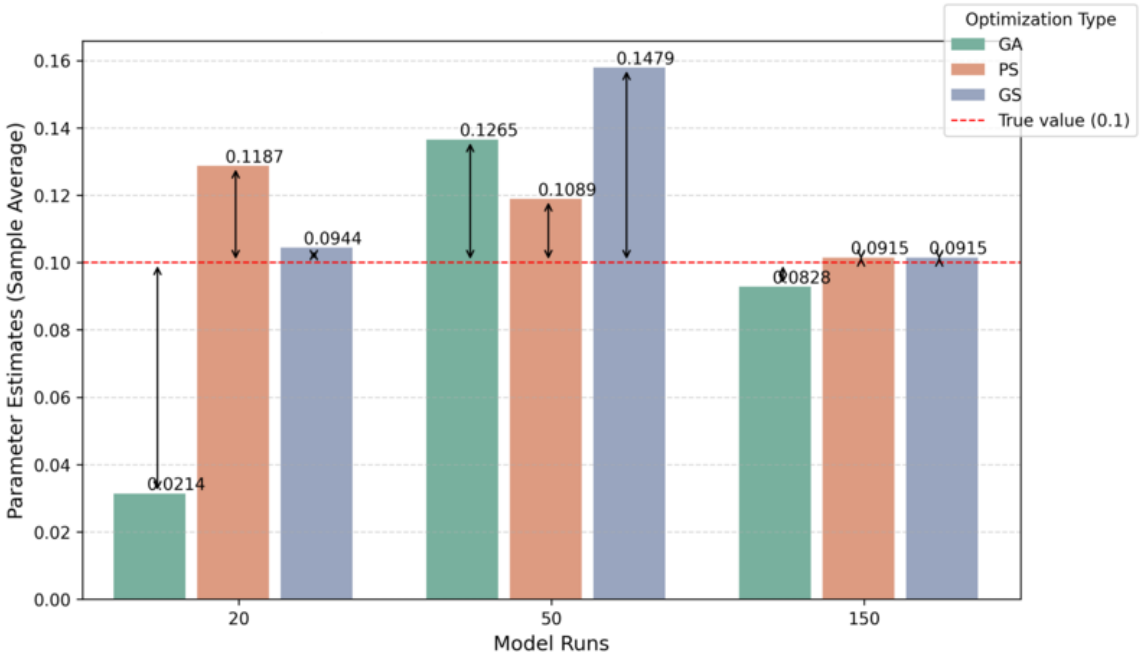
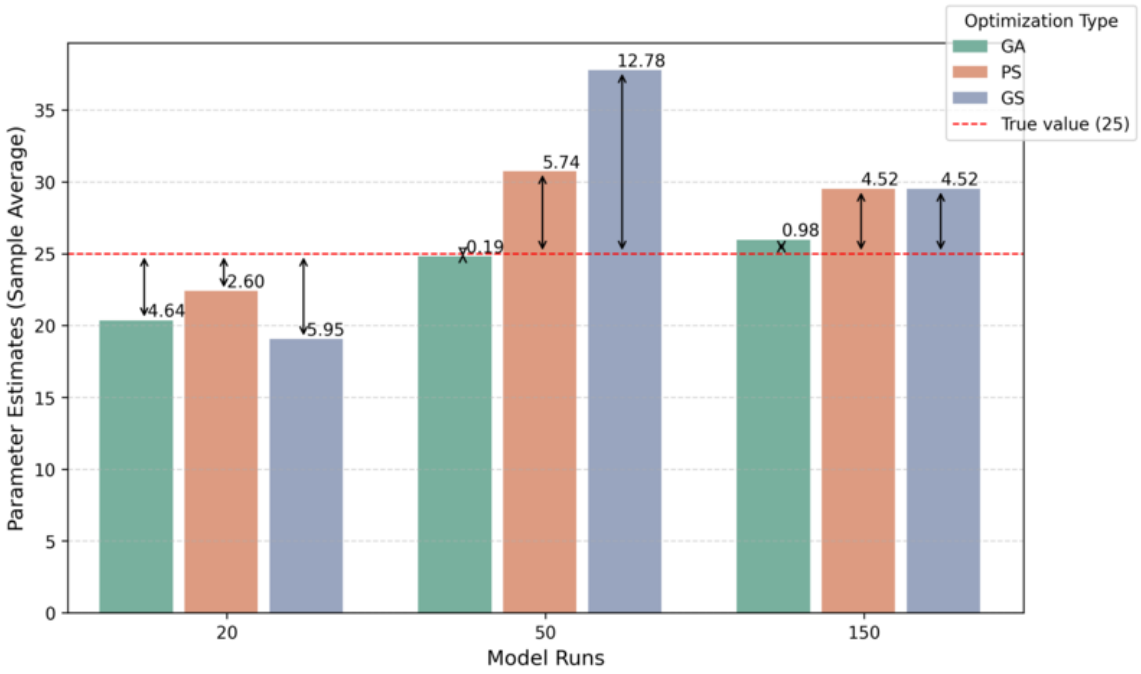


Figure 6: MCS Estimated Bias Results - Prior Strength (Z)

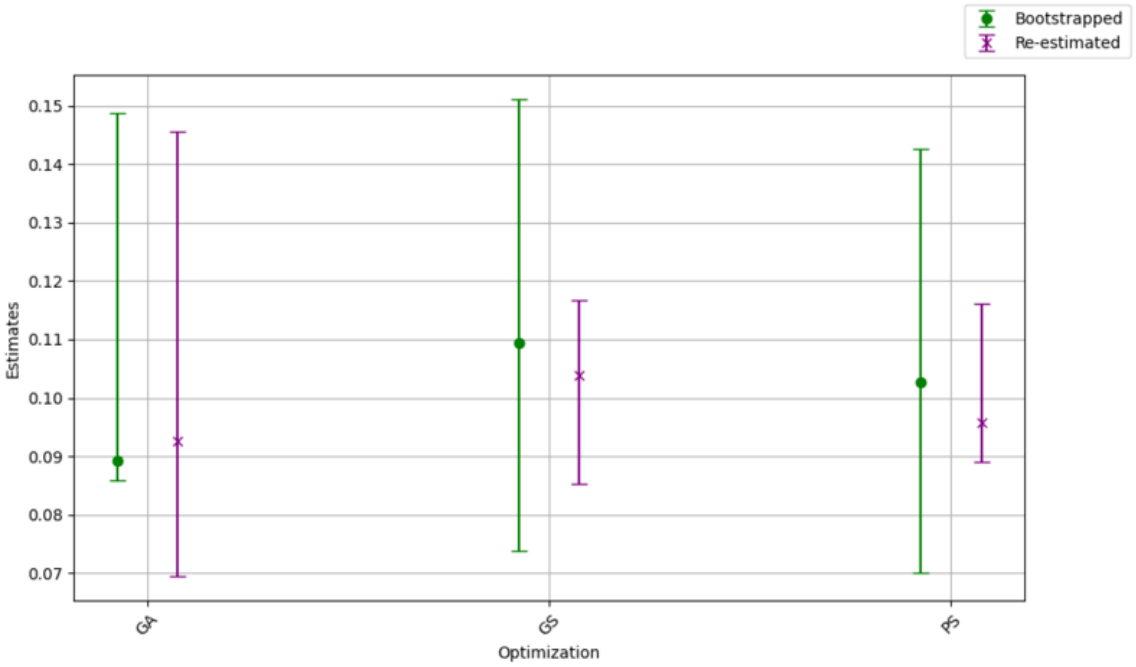


5.6 We see that while the bias in R seems to be decreasing in runs, the relationship between the bias in our estimates of Z and run number is not as clear. One possible way to remedy this is to simply subtract the recovered bias from our best estimates of Z we get on data.

5.7 **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 come 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.

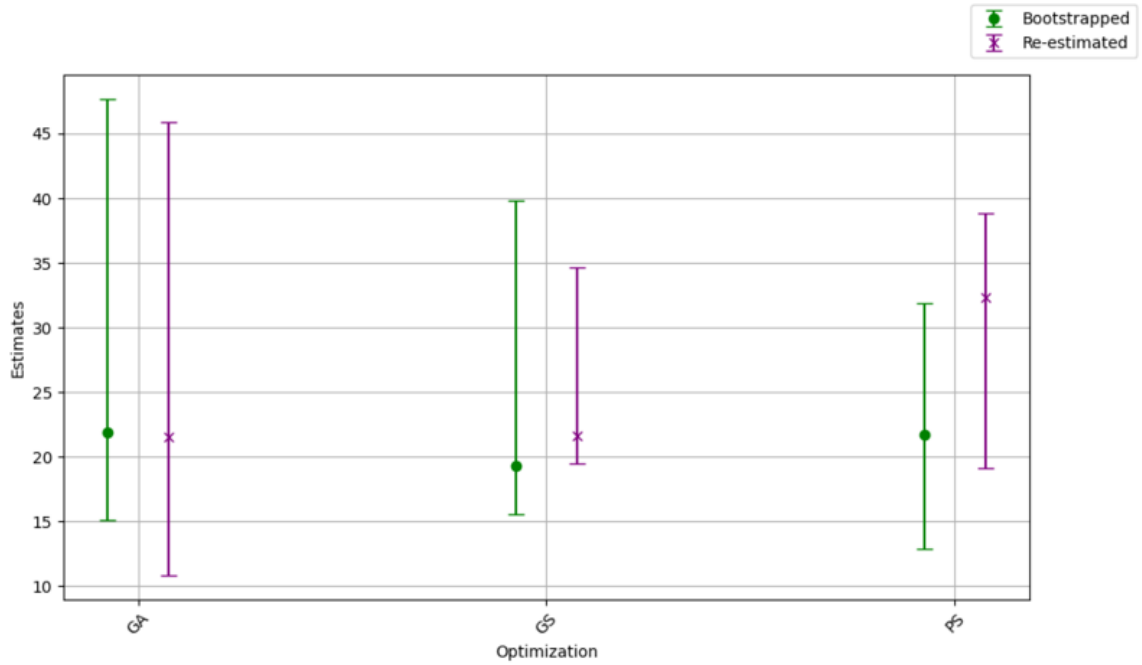
Figure 7: MCS Imprecision Source Comparison - Recency Bias (R)



5.8 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.

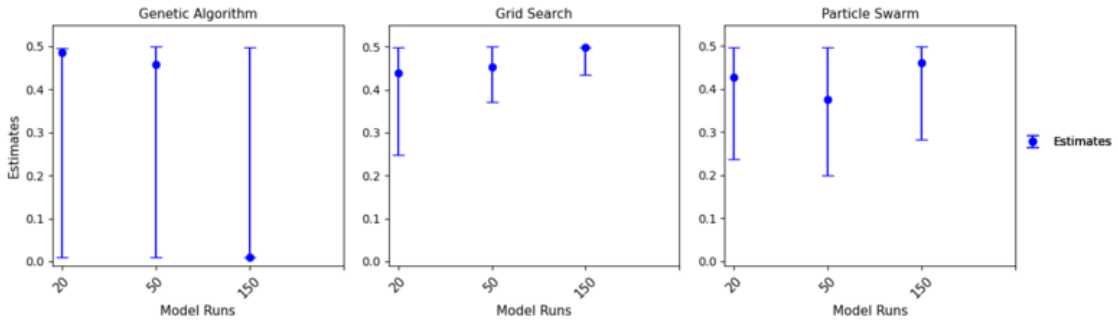
Figure 8: MCS Imprecision Source Comparison - Prior Strength (Z)



5.9 Once again, we see (above) the GA results spans 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.

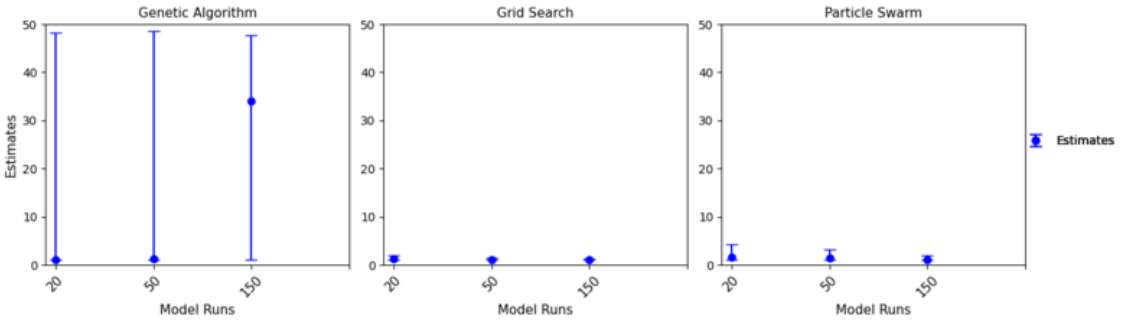
5.10 **Data Results:** At this point, analysts using GS or PS as their $search(.)$ process with these specifications could proceed with estimation of the real data with greater confidence. Again, the process of finding best fitting parameters θ^* and CIs is precisely the same as was done in the Monte 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)



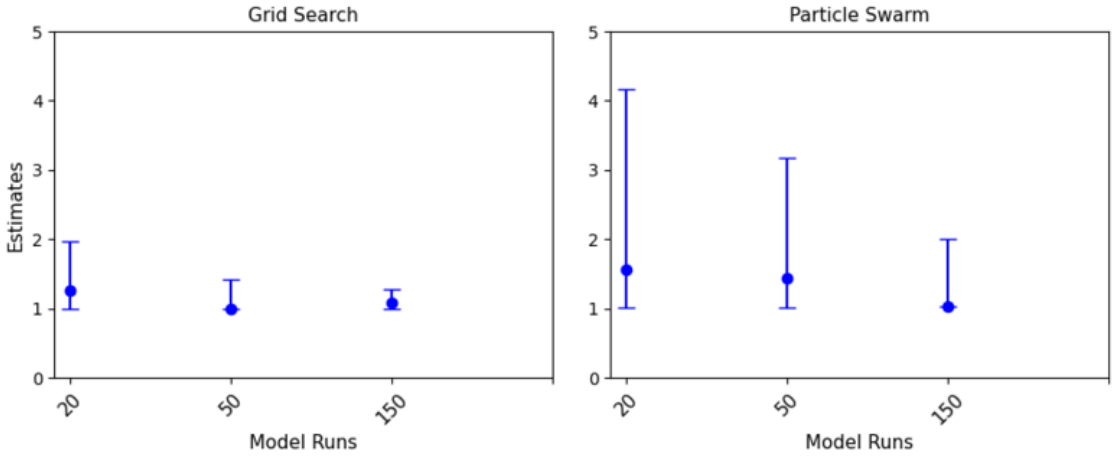
5.11 As indicated by our simulations, it seems Grid Search (GS) and Particle Swarm Optimization (PSO) at the highest level of runs (r=150) can produce much more precise estimates of R. Further, it seems both GS and PSO return similar estimates of R, with GS best estimating R = 0.50 with CI = $[0.43,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

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



5.12 Once again, Grid Search (GS) and Particle Swarm Optimization (PSO) produce fairly precise estimates or 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



Example Application 2 - Information Diffusion Model

Application Set Up

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

6.2 **The Data:** We utilize one data set used in Rand et al. (2015) on the diffusion of information pertaining to Hurricane Sandy on Twitter. First, a snowball network sample of 15,000 users was collected to construct the network. Next, using the Twitter API, tweet 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 is then done to identify the first time an individual mentioned a keyword in a tweet or a retweet.

6.3 TODO How to block bootstrap

6.4 **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.

6.5 In this model, each agent (a.k.a nodes) either knows about Hurricane Sandy or not. When an agent first learns about Sandy, we say they are *activated*. Once an agent is activated, they remain activated forever. When the model is initialized, a single agent is chosen to start activated. 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**. Agents have a probability **p** of learning from an outside source each round and a probability **q** to learn from their neighbors if at least one of their neighbors was activated in the previous time step. So p and q make up the vector of parameters θ which we want to estimate on our data.

6.6 **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 p and q, 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 just a sample of a much larger social network both online and offline. One particularly strong assumption is the assumption 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.

6.7 **On Estimating Best Fitting Parameters:**

6.8 **On Bootstrapping:**

6.9 **On Monte-Carlo Simulation:**

Results

6.10 **MCS Results for Best Fits and CIs:**

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

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

Outlook and Conclusion

7.1 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. 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 under different estimation configurations. This underscores the danger of relying on standard methods without verifying their suitability for a given application.

7.2 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

Leonardo Bargigli. Chapter 8 - econometric methods for agent-based models. In Mauro Gallegati, Antonio Palestrini, and Alberto Russo, editors, *Introduction to Agent-Based Economics*, pages 163–189. Academic Press, 2017. ISBN 978-0-12-803834-5. doi: https://doi.org/10.1016/B978-0-12-803834-5.00011-4. URL https://www.sciencedirect.com/science/article/pii/B9780128038345000114.

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.

Russel Davidson and James G. MacKinnon. Bootstrap inference in econometrics. *Wiley-Blackwell: Canadian Journal of Economics*, 2002.

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.

William H. Greene. *Econometric Analysis*. Pearson Education, eighth edition, 2017. ISBN 0-13-446136-3.

R.H. Hoyle. *Handbook of Structural Equation Modeling*. Guilford Publications, 2012. ISBN 9781462504473. URL https://books.google.com/books?id=qC4aMfXL1JkC.

James G. MacKinnon. Bootstrap methods in econometrics. *Economic Record*, 82(s1):S2–S18, 2006. doi: https://doi.org/10.1111/j.1475-4932.2006.00328.x.

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.