JASSS
THE JOURNAL OF ARTIFICIAL SOCIETIES AND SOCIAL SIMULATION

Monte-Carlo Tests for Identification and Validation of Stochastic Agent-Based Models

[Anonymised review copy]

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 through the conventional inferential frameworks that underpin most econometric practice.

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 in some cases. 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.

Doi: 10.18564/jasss.xxxx

Introduction

Although computational models have traditionally been used to demonstrate proof-of-principle results (as in (Schelling 1971)), it has become increasingly common and desirable to employ agent-based models (ABMs) for direct structural estimation. *Structural estimation*, like conventional econometric estimation using regression, seeks to recover parameter values that, when used within the model, produce simulated outcomes that best match empirical observations, often in terms of moments or distributions of outcome variables. Unlike reduced-form estimation, which typically relies on a (usually linear) representation of numerous variables and their interactions, structural estimation proceeds by taking as given a specific, often more restrictive functional form, usually motivated by prior knowledge of the underlying system. Structural estimation is widely used in many fields, including economics and industrial organization, particularly for dynamic models in which behavior is governed by explicit decision rules and equilibrium conditions.

Structural estimation relies on the validity of the underlying structural assumptions. A key strength of ABMs is their ability to represent rich behavior, heterogeneity, and complex interactions without imposing strong requirements for mathematical tractability. Pairing ABMs with structural estimation allows one to accommodate this richness. At the same time, this flexibility often results in highly nonlinear models, making analytical solutions for best-fitting parameters rarely feasible.

Moreover, standard concerns such as identification, bias, and estimator precision are more difficult to establish without additional assumptions. If direct estimation of ABMs is to be taken seriously, empirical results must therefore be preceded by systematic validation of estimator performance. This, in turn, requires inference procedures that are flexible and model-agnostic, avoiding the imposition of restrictive structure *a priori*.

In computer science and machine learning, model assessment typically emphasizes out-of-sample performance, with a clear separation between training and evaluation data. These approaches prioritize predictive generalization. By contrast, empirical economic practice has traditionally focused on in-sample estimation and inference, with uncertainty commonly quantified using confidence intervals. Confidence intervals play a central role in economic research because they are used to assess whether estimated parameters are empirically meaningful and statistically distinguishable from zero.

In broad terms, out-of-sample approaches evaluate how well the full set of model parameters predicts outcomes, whereas in-sample inference speaks to the relative importance and interpretability of individual parameters within the context of the model. For example, a parameter vector that performs well out-of-sample might suggest, from a policy perspective, that outcomes could be improved by aligning real-world conditions with those parameter values. In contrast, in-sample methods may reveal that, conditional on the parameter space explored, some parameters contribute little to the objective of interest. If adjusting certain parameters is costly, this information is crucial for weighing feasible policy alternatives.

Importantly, in-sample inference is complementary to out-of-sample evaluation and provides a controlled setting for studying the statistical properties of estimators. Unlike in many other fields, economics rarely reports empirical results without accompanying confidence intervals or p-values, making reporting such values, from a very practical perspective, essential to nearly all applied work. For agent-based models (ABMs), however, the standard distributional and structural assumptions underlying conventional in-sample inference are unlikely to hold, creating a gap between estimation practice and inferential credibility. Rather than abandoning in-sample methods altogether, this paper demonstrates how Monte Carlo simulation (MCS), a well-established computational approach widely used in econometrics, can be employed to address these inferential challenges.

Monte Carlo simulation is a general computational technique for approximating the sampling distribution of an estimator by repeatedly simulating data from a model under known or estimated parameter values and recomputing statistics of interest across simulations. In the context of highly nonlinear models, MCS provides a flexible approach for conducting in-sample inference when standard asymptotic results are unavailable.

Using MCS, we detail a general, model-agnostic framework for assessing key inferential properties of ABMs and their estimators, including parameter identifiability, finite-sample bias, and estimator precision. In particular, we measure imprecision using confidence intervals constructed using a particular Monte Carlo method called bootstrapping. This framework further allows researchers to examine how estimator performance depends on practical modeling choices, such as optimization algorithms, the number of model runs, and the specification of fitness functions. Such analysis is essential for determining whether parameters can be reliably recovered and for clarifying the conditions under which estimated parameters meaningfully correspond to real-world counterparts, as we demonstrate in two illustrative 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 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. In Sections and , we apply these MCS-based tests to the two example problems mentioned above and interpret their results. Finally, we conclude in Section .

Literature Review

Outside of economics, where the use of agent-based models (ABMs) are more commonplace, applications often emphasize calibration and out-of-sample fit, with comparatively little attention paid to identification, estimator precision, or the sources of estimation error. A substantial literature focuses on the problem of tuning high-dimensional parameter spaces in ABMs using automated search procedures, including genetic algorithms and other heuristic optimization techniques (Calvez and Hutzler 2006; Chica et al. 2016). For example, Stonedahl and Rand (2012) study how alternative error measures affect calibration outcomes and out-of-sample performance, demonstrating that estimated parameters can be highly sensitive to the choice of fitness function. While this work and related studies establish that ABMs can be calibrated to match empirical patterns, evaluation is typically framed in terms of predictive accuracy or goodness of fit rather than the formal statistical properties of the resulting estimators.

In contrast, this paper builds on these calibration-based approaches by grounding ABM estimation more explicitly in the econometric tradition of in-sample methods which, as noted above, can be used in tandem with out-of-sample methods. We formalize the use of Monte Carlo simulation for ABMs with the specific aim of evaluating estimator properties such as precision, robustness, and sources of estimator imprecision, tasks for which simulation-based methods are particularly well suited. Our focus is not only on whether a model can be calibrated successfully, but on understanding what can be inferred from the estimated parameters and how reliable those inferences are.

Conventional estimation practice in economics differs in important ways from estimation approaches commonly used in fields such as machine learning or evolutionary computation. Empirical analyses in economics typically report formal measures of statistical uncertainty, including confidence intervals, p-values, or critical values, and place substantial emphasis on well-understood regression and estimation frameworks. Robustness analysis, both with respect to model specification and estimation procedure, is treated as a central component of credible empirical work. As a result, discussions of empirical findings often focus as much on identification strategies and robustness checks as on point estimates or predictive performance alone. This emphasis reflects a broader concern with interpretability and policy relevance, rather than calibration accuracy per se.

Within the econometric simulation literature, there has been extensive development of methods for estimating nonlinear and stochastic models. Many such approaches, however, require the model to be fully characterized by a system of equations, often subject to differentiability or smoothness conditions, as in maximum likelihood estimation, or rely on restrictive assumptions about the distribution of unobserved shocks. For many

ABMs, these requirements are difficult or impossible to satisfy, as model dynamics typically involve substantial stochasticity and iterative, algorithmic components that cannot be compactly represented in closed-form equations.[1] These challenges have been recognized since the early simulation literature, which emphasized both the promise and the methodological difficulties inherent in digital system simulation (Conway et al. 1959).

At the same time, Monte Carlo simulation in economics (contrasted with the meaning of "Monte Carlo" in agent-based modeling) has long played an important role in econometrics for studying the finite-sample behavior of estimators and test statistics, particularly in settings where standard functional form assumptions are unlikely to hold (Davidson and MacKinnon 2002; MacKinnon 2006). The use of Monte Carlo simulation as a tool for evaluating estimator precision and decomposing sources of estimation error in data-driven ABMs remains relatively underexplored. The closest analogue might be Miller (1998), in which active nonlinear tests are designed to probe the structural robustness of complex simulation models by deliberately searching for parameter configurations that expose model weaknesses. While these approaches are not framed as estimation procedures, they highlight the value of systematic simulation-based diagnostics for understanding the behavior of complex models beyond simple goodness-of-fit measures.

Our estimation approach relies on the Simulated Method of Moments (SMM), a flexible and widely applicable framework for parameter estimation in simulation-based models. SMM was introduced by McFadden (1989) and is summarized in several econometric texts, including the classic treatment of simulation-based inference by Gourieroux and Monfort (1996). We adapt this framework to the ABM setting, using Monte Carlo methods to construct confidence intervals and to additionally explore estimator properties, and discuss practical considerations related to the choice of objective functions, run aggregation, and search algorithm used for optimization. Additional details of the implementation are provided in Zosh et al. (2025).

Finally, we introduce a novel diagnostic test designed to decompose the sources of estimation imprecision in simulation-based models (Test 4 in Section). To our knowledge, no existing procedure provides a comparable method for isolating the contributions of simulation noise, optimization error, and model structure to the overall estimator variability.

ABM Estimation Overview

This section provides a description of the our Monte Carlo estimation process, including an abridged overview of relevant ideas and components. We briefly discuss estimating ABM parameters, computing confidence intervals, and interpreting parameter estimates.[2]

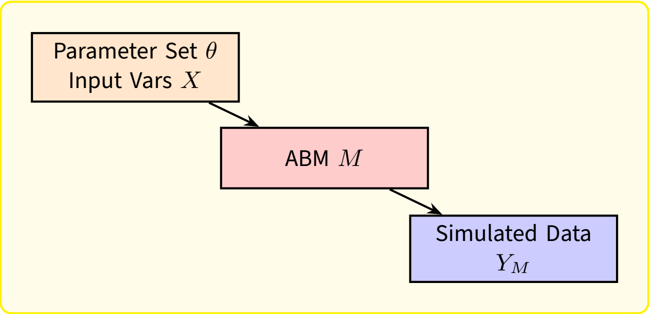


Figure 1: Simulated Data Generation using an ABM

Bringing an Agent-based Model to Data

Figure 1 depicts an ABM M which takes parameters θ and returns model output $M(\theta, X) \rightarrow Y_M$. Depending on the specification, M may also take exogenous variables X from a dataset D as inputs, which may be utilized in the model in each period or serve solely to initialize model conditions. Depending on the context, the simulated data Y_M might be one simulated output of the model (i.e. one "run") or multiple runs, providing a distribution of outcomes, where each output may provide one or more outcomes of interest.

[1] As an illustration, imagine Schelling's classic segregation model (Schelling 1971) as a system of equations.

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

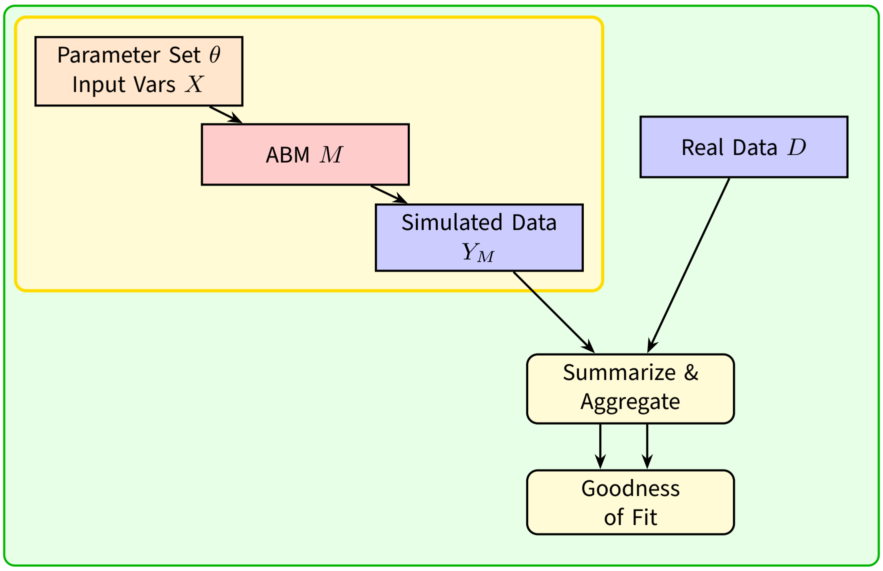


Figure 2: Fit of Parameter Set θ

Figure 2 depicts the fit of parameters θ to data. Note that the entirety of Figure 1 is contained within Figure 2. The Real Data D are compared to the simulated data Y_M via some "Goodness of Fit" metric. Goodness of fit can be evaluated by many different methods, for example mean squared differences between the simulated and real data.

We define Goodness of Fit process via three functions: a *summary function* $S(\cdot)$, an *aggregation function* $Agg(\cdot)$, and a *fitness function* $fit(\cdot)$

A *summary function* $S(\cdot)$ produces summary statistics of output Y_M from the model M (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 $Z(Y_M, \theta)$ and $Z(Y_D)$ respectively.

An *aggregation function* $Agg(\cdot)$ aggregates summarized output across r runs of M. 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 $\bar{Z}(Y_M, \theta, r)$ which, in our examples, simply calculates the average of each summary statistic generated across runs. This is given by:

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

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

A *fitness function* $fit(\cdot)$ establishes how to compare the aggregated output of M, $\bar{Z}(Y_M, \theta, r)$ 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:

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

where,

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

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:

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

where,

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

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:

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

Finding Best-fitting Parameters

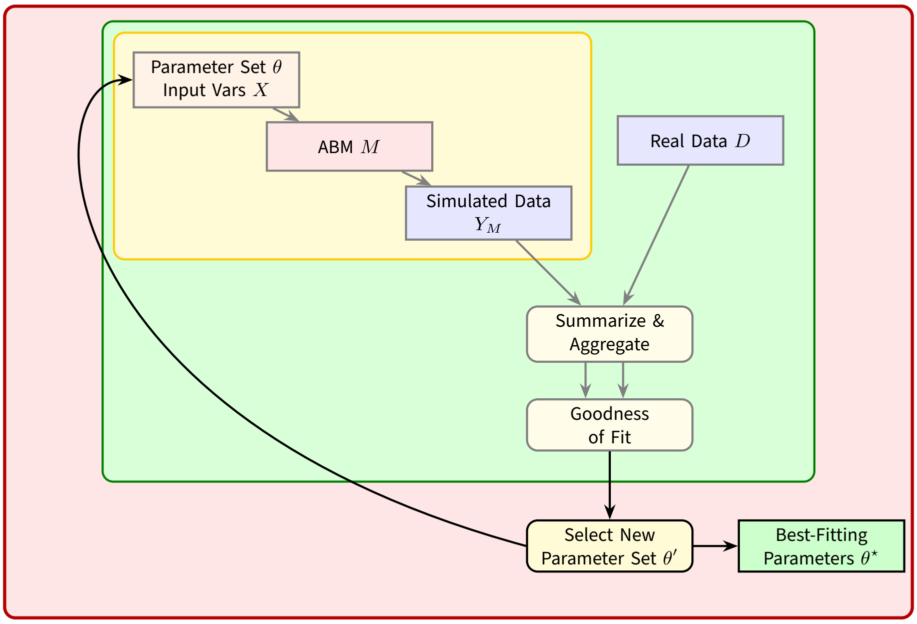


Figure 3: Finding Best-Fitting Parameters $\theta^\star$

Figure 3 depicts finding best-fitting parameters $\theta^\star$. Note that the entirety of Figure 2 is contained within Figure 3. The best-fitting parameters are a set of parameters which, when used in model M, produce output Y_M which best emulates the the salient features of Y_D according to the Goodness of Fit metric described above.

An *optimization technique* $search(\cdot)$ searches for parameters $\theta^\star$ which together maximize our fitness function $fit(\cdot)$, searching the space of possible parameters guided by its own search parameters δ. In Figure 3, search process is illustrated by the "Select new Parameter Set" step. In this paper, we explore outcomes using three different optimization technique specifications: Grid Search, the Genetic Algorithm, and Particle Swarm Optimization, but any optimization technique could be used.

With operations S, Agg, fit, and $search$ defined, the best-fitting parameters $\theta^\star$ can be estimated by doing the following nested operation:

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

where

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

This process searches for the set of parameters $\theta^\star$ which maximizes the measure of fitness between our aggregated summarized ABM output and our summarized observations from data.

Bootstrapping Confidence Intervals

Figure 4 depicts the process of bootstrapping standard errors. Note that the entirety of a modified Figure 3 is contained within Figure 4. (The modification is that the parameters are fit a bootstamped sample of the real data

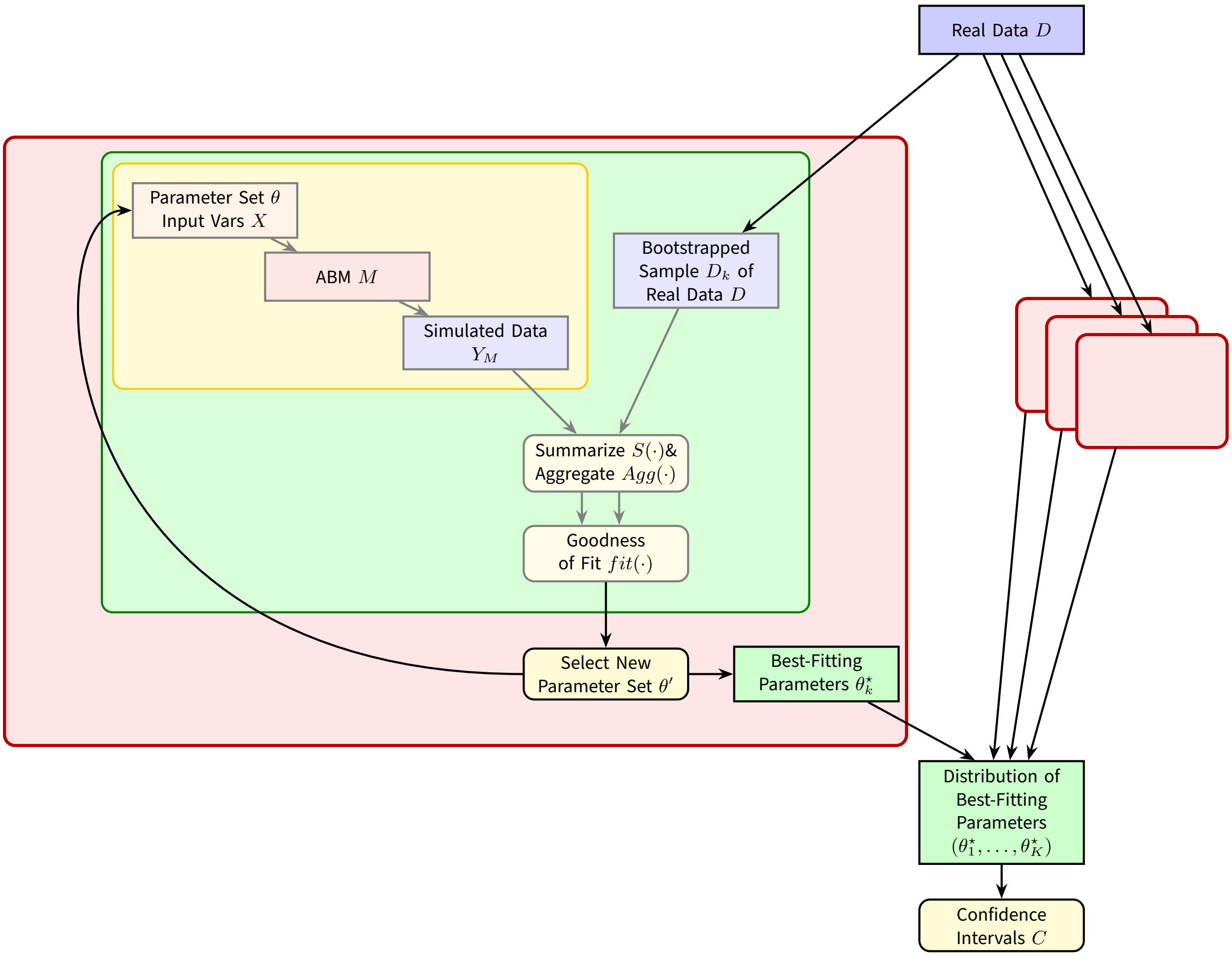

Real Data D
Parameter Set θ
Input Vars X
ABM M
Simulated Data Y_M
Bootstrapped Sample D_k of Real Data D
Summarize $S(\cdot)$& Aggregate $Agg(\cdot)$
Goodness of Fit $fit(\cdot)$
Select New Parameter Set θ'
Best-Fitting Parameters $\theta_k^\star$
Distribution of Best-Fitting Parameters $(\theta_1^\star, \ldots, \theta_K^\star)$
Confidence Intervals C

Figure 4: Bootstrapping a Distribution of Best-Fitting Parameters

instead of the real data itself.) In Figure 4, the real data D is sampled via a so-called bootstrapping process, and then best fitting parameters are found (as shown in Figure 3). This is done many times for different bootstrapped samples, as illustrated by the small red rectangles to the right, meant to invoke copies of the modified Figure 3. This process creates a distribution of parameter estimates $(\theta_1^\star, \ldots, \theta_K^\star)$.

The idea behind bootstrapping is that the observed real data D is only one possible data set that could have been observed from the unobserved true underlying data generating process. Bootstrapping simulates a sample of possible "real data" sets from the observed data D. If the bootstrapping process is successful, this will provide an accurate distribution of the values that $\theta^\star$ could have taken on. This allows us to ask questions like "Is this estimated value for θ significantly different from zero?" That is, this distribution The distribution of parameter estimates $(\theta_1^\star, \ldots, \theta_K^\star)$ can be used to calculate critical values or confidence intervals for these estimates $\theta^\star$. These critical values are useful both for hypothesis testing and for establishing how sensitive parameter estimates are to sampling variation.

There are a number of ways to construct bootstrapped samples for computational models. In our examples, we generate the bootstrapped samples using block-bootstrapping. In summary, our panel data D is broken into independent blocks, which are then resampled to generate new, simulated data sets. The best-fitting parameters for each of these simulated datasets provides the distribution of best-fitting parameter estimates $(\theta_1^\star, \ldots, \theta_K^\star)$. This distribution is used to construct confidence intervals for each parameter.

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 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 $\theta^\star$ 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.

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 $\theta^\star$ (see equation 7), even when fitting the same data, due to the stochastic nature of both $search(\cdot)$ and our model M. Thus, we can think of $search(\cdot)$, which aims to fit our model parameters to data, as returning the estimated parameters $\theta^\star$ 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 $\theta^\star$ 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 is precisely the true DGP, DGP_{Data}. Next, let us choose some parameters θ_{ch} for which our model is well defined. Given these two, we could then use our model M and the chosen parameters θ to generate a simulated dataset by recording the model output Y_M.

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

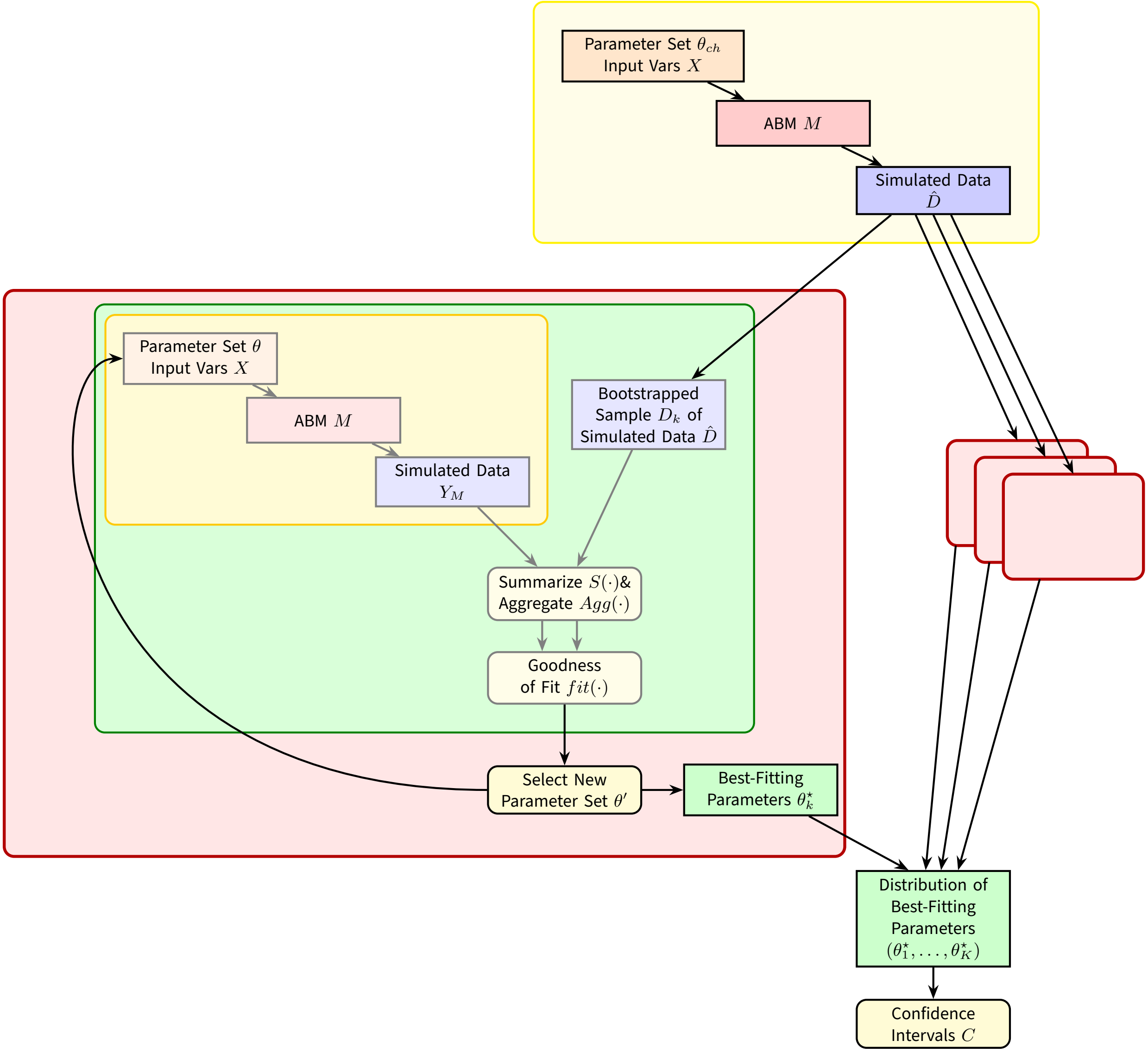


Figure 5: Bootstrapping a Distribution of Best-Fitting Parameters for Simulated Data

Importantly, these data were generated using parameter values θ_{ch} that are known, because we chose them. As seen in Figure 5, the Monte Carlo process will produce estimates and confidence intervals of our chosen parameters θ_{ch}. This will allow us to learn quite a bit about the properties of 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: Estimate Accuracy

The first and arguably most important test of the model is to establish whether or not our estimation technique can return estimates $\theta^\star$ which are reasonably close to the true known values we have chosen θ_{ch}.

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

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

Then we simply compare each value in θ_{ch} and $\theta^\star$ 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 and .

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 , but use our simulated dataset $\hat{D}$ instead of our real data D. Formally:

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:

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

Find K Best Fits $\theta_k^\star$ for Each Using the $\hat{D}_k$

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

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

Construct the Critical Value(s)

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

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

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.

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:

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

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_{ch} and the average of parameter estimate θ^{i*}.

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

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 $Agg(M, r)$ 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:

Find K Best Fits $\theta^{\star}_{k}$ 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.

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_M, \theta, r), Z(Y_{\hat{D}_k})), \delta) \rightarrow \theta^{\star}_{k} \tag{15}$$

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 .

$$\hat{C}_i = [\theta^{\star}_i + \varepsilon^i_{mth}, \theta^{\star}_i + \varepsilon^i_{nth}] \tag{16}$$

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.

Application: Repeated Prisoner's Dilemma with Learning Agents

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 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.k.a DGP_M) 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 $\bar{Z}(Y_M, \theta, r)$ 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:

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

where,

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

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 , 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 . 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.

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 6: Monte Carlo Simulation Results - Recency Bias (R)

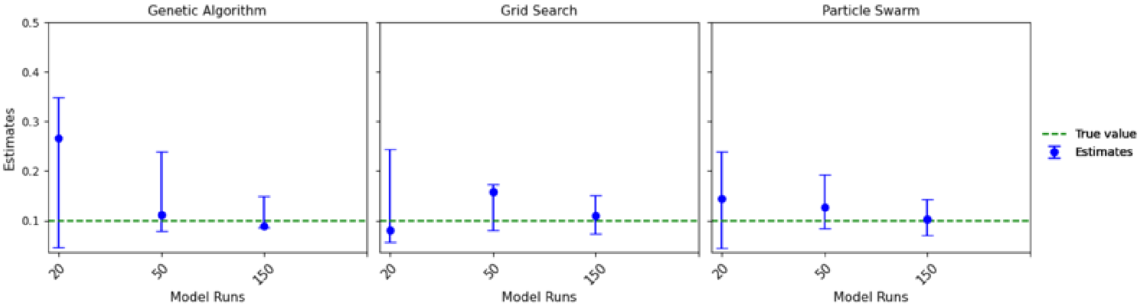


Table 2: Recency bias, simulated data (Real Data = FALSE)

Optimization	Model Runs	True Value	Estimate	Lower CI	Upper CI
Genetic Algorithm	20	0.1000	0.2670	0.0466	0.3487
Genetic Algorithm	50	0.1000	0.1120	0.0781	0.2389
Genetic Algorithm	150	0.1000	0.0893	0.0858	0.1487
Grid Search	20	0.1000	0.0812	0.0568	0.2443
Grid Search	50	0.1000	0.1586	0.0810	0.1733
Grid Search	150	0.1000	0.1094	0.0740	0.1511
Particle Swarm	20	0.1000	0.1445	0.0447	0.2392
Particle Swarm	50	0.1000	0.1263	0.0839	0.1930
Particle Swarm	150	0.1000	0.1026	0.0701	0.1427

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 7: Monte Carlo Simulation Results - Prior Strength (Z)

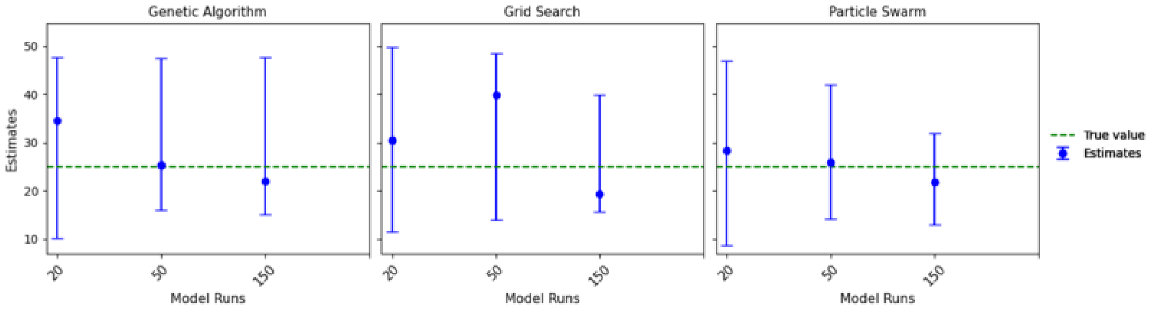


Table 3: Strength of prior, simulated data (Real Data = FALSE)

Optimization	Model Runs	True Value	Estimate	Lower CI	Upper CI
Genetic Algorithm	20	25.0000	34.6113	10.1135	47.6222
Genetic Algorithm	50	25.0000	25.3231	15.9649	47.4708
Genetic Algorithm	150	25.0000	21.8993	15.0731	47.6759
Grid Search	20	25.0000	30.5254	11.5570	49.7358
Grid Search	50	25.0000	39.8080	13.9932	48.4771
Grid Search	150	25.0000	19.2986	15.5793	39.8652
Particle Swarm	20	25.0000	28.2696	8.6361	46.9777
Particle Swarm	50	25.0000	25.8337	14.1869	41.9774
Particle Swarm	150	25.0000	21.7424	12.9250	31.8990

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

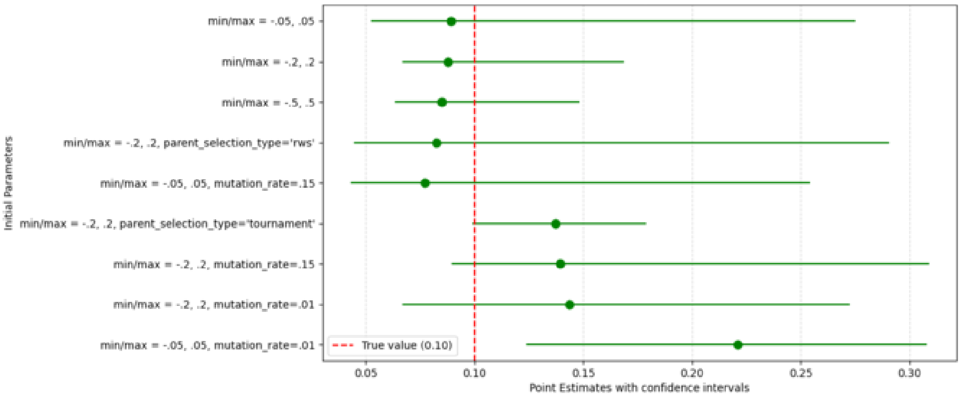
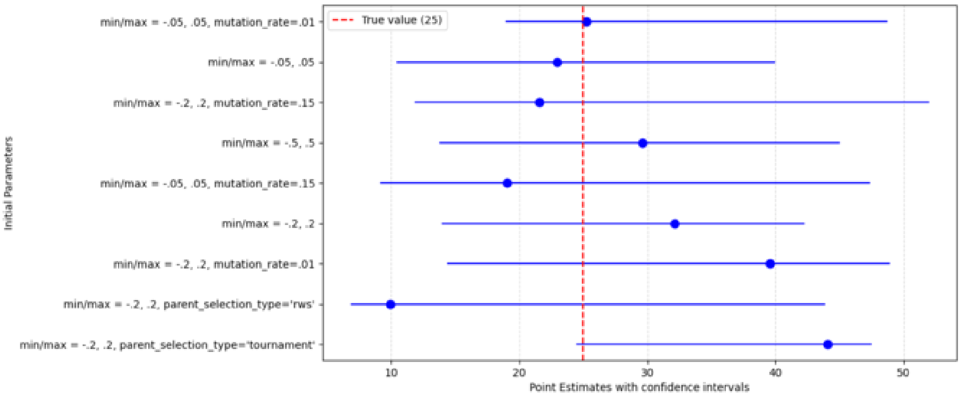


Figure 9: MCS GA Results Using Alternate δs - Prior Strength (Z)



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 10: MCS Estimated Bias Results - Recency Bias (R)

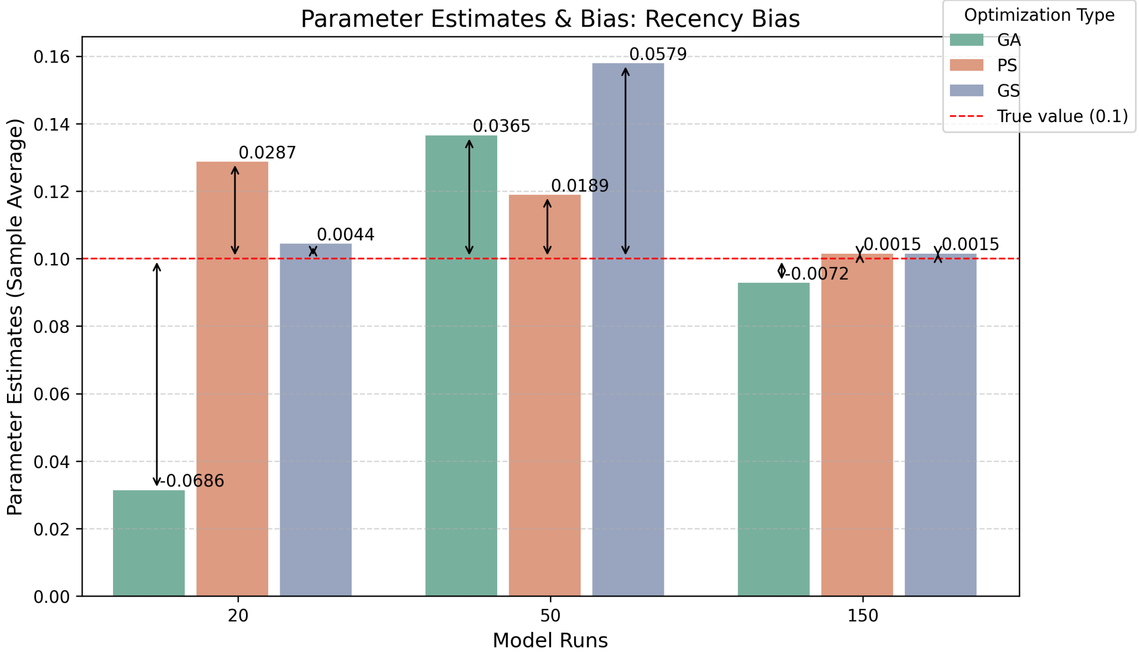


Table 4: Recency Bias: Average and Bias by Model Runs and Optimization Type

Model Runs	Optimization Type	R Average	R Bias
20	Genetic Algorithm	0.0314	-0.0686
20	Particle Swarm	0.1287	0.0287
20	Grid Search	0.1044	0.0044
50	Genetic Algorithm	0.1365	0.0365
50	Particle Swarm	0.1189	0.0189
50	Grid Search	0.1579	0.0579
150	Genetic Algorithm	0.0928	-0.0072
150	Particle Swarm	0.1015	0.0015
150	Grid Search	0.1013	0.0013

Figure 11: MCS Estimated Bias Results - Prior Strength (Z)

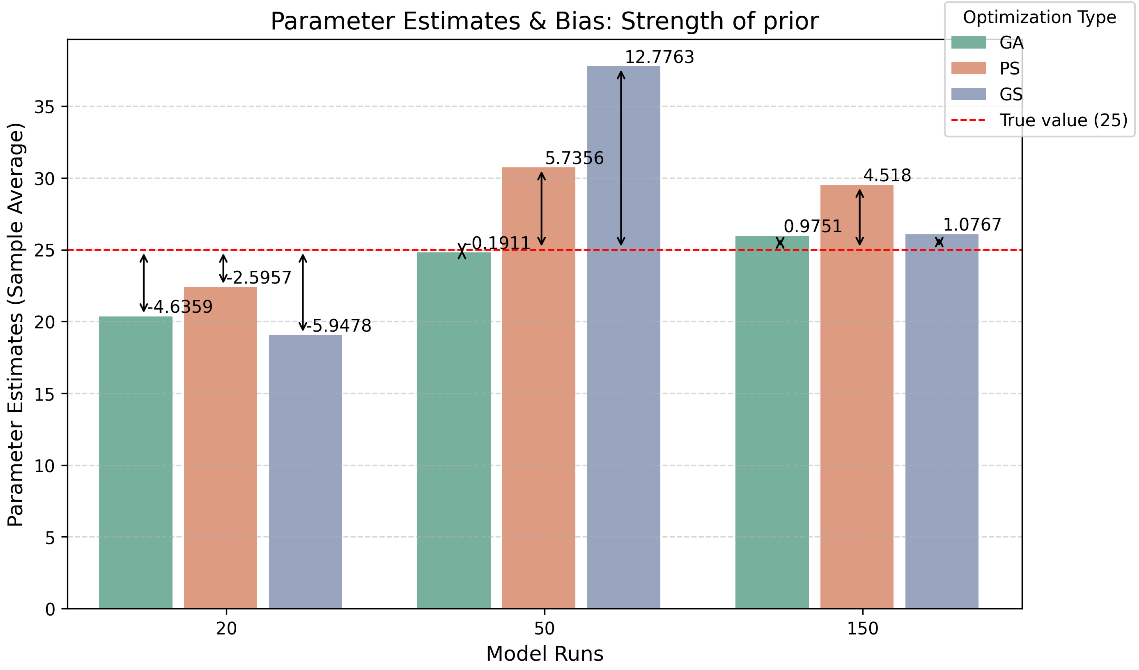


Table 5: Strength of Prior: Average and Bias by Model Runs and Optimization Type

Model Runs	Optimization Type	Z Average	Z Bias
20	Genetic Algorithm	20.3641	-4.6359
20	Particle Swarm	22.4043	-2.5957
20	Grid Search	19.0522	-5.9478
50	Genetic Algorithm	24.8089	-0.1911
50	Particle Swarm	30.7356	5.7356
50	Grid Search	37.7763	12.7763
150	Genetic Algorithm	25.9751	0.9751
150	Particle Swarm	29.5180	4.5180
150	Grid Search	26.0767	1.0767

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 12: MCS Imprecision Source Comparison - Recency Bias (R)

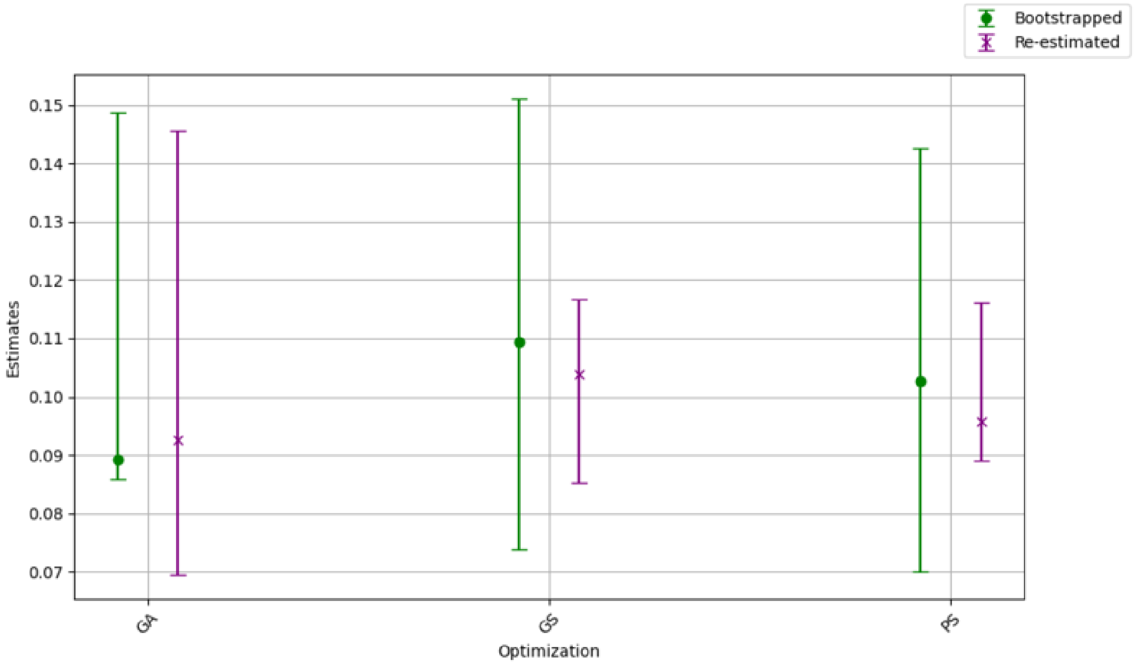


Table 6: Recency bias — All Reestimates, sorted by Optimization

Reestimates	Optimization	Rounds	Estimate	Lower CI	Upper CI
False	GA	100	0.0927	0.0695	0.1455
True	GA	100	0.0893	0.0858	0.1487
False	GS	5	0.1038	0.0852	0.1168
True	GS	5	0.1094	0.0739	0.1511
False	PS	100	0.0957	0.0890	0.1160
True	PS	100	0.1026	0.0701	0.1427

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 13: MCS Imprecision Source Comparison - Prior Strength (Z)

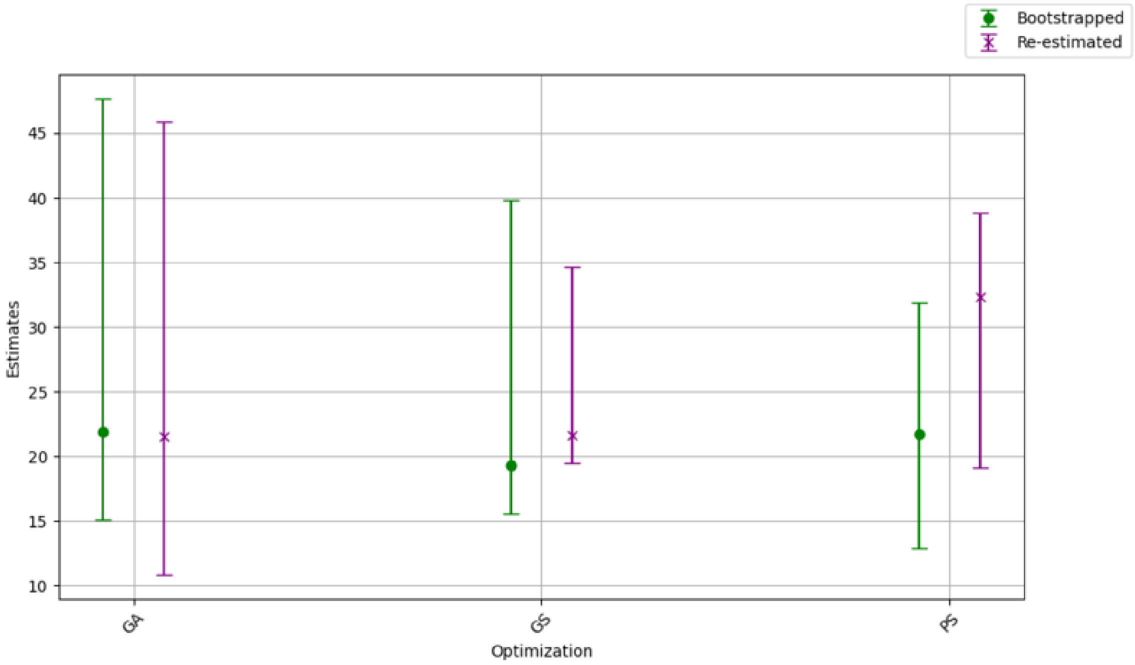


Table 7: Strength of prior — All Reestimates, sorted by Optimization

Reestimates	Optimization	Rounds	Estimate	Lower CI	Upper CI
False	GA	100	21.5380	10.8417	45.9320
True	GA	100	21.8993	15.0731	47.6759
False	GS	5	21.6192	19.4507	34.6826
True	GS	5	19.2986	15.5793	39.8652
False	PS	100	32.3090	19.0927	38.8831
True	PS	100	21.7424	12.9250	31.8990

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 $\theta^{\star}$ 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 both in the figure and table below.

Figure 14: Estimates on Data - Recency Bias (R)

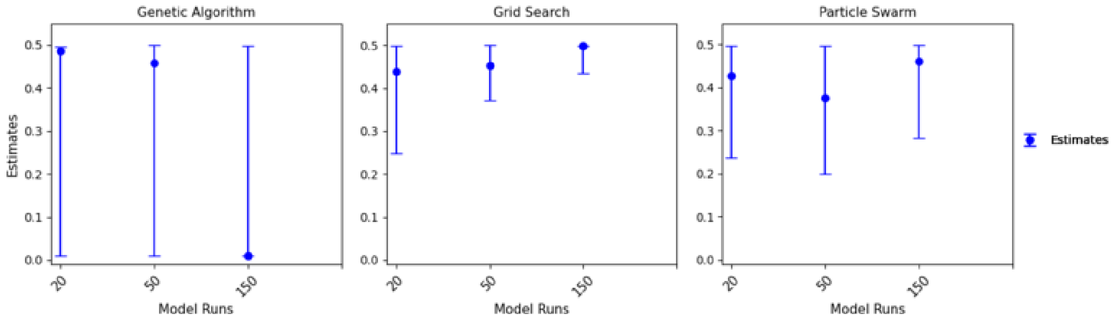
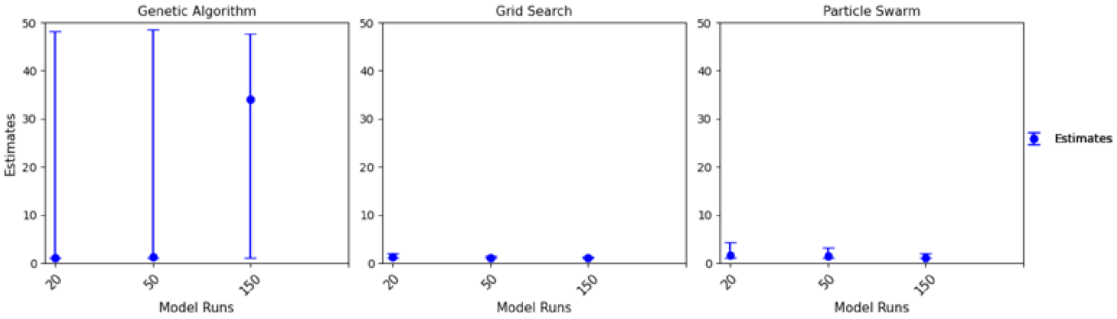


Table 8: Recency bias, real data (Real Data = TRUE)

Optimization	Model Runs	Estimate	Lower CI	Upper CI
Genetic Algorithm	20	0.4858	0.0100	0.4963
Genetic Algorithm	50	0.4569	0.0100	0.4991
Genetic Algorithm	150	0.0100	0.0100	0.4975
Grid Search	20	0.4393	0.2482	0.4991
Grid Search	50	0.4520	0.3711	0.5000
Grid Search	150	0.4987	0.4348	0.4991
Particle Swarm	20	0.4279	0.2380	0.4968
Particle Swarm	50	0.3762	0.2001	0.4963
Particle Swarm	150	0.4602	0.2821	0.4981

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 15: Estimates on Data - Prior Strength (Z)

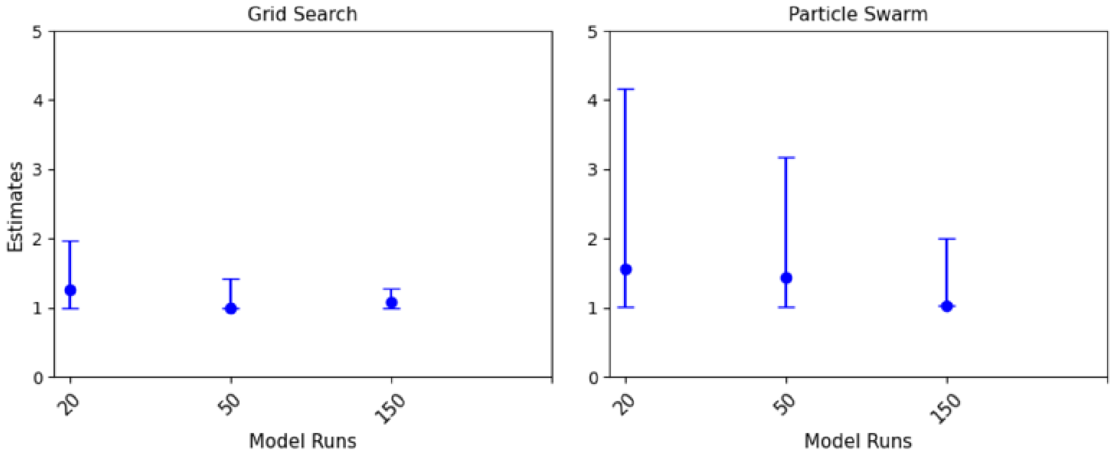


Once again, Grid Search (GS) and Particle Swarm Optimization (PSO) produce fairly precise estimates of Z at (r=150) while the Genetic Algorithm seems to produce CIs spanning nearly all possible values Z is allowed to take. It also appears that GS and PSO are in relative agreement, with GS best estimating Z = 1.04 with CI = $[1.02, 2.01]$ while PSO returns Z = 1.08 with CI = $[1.00, 1.27]$. Below we include the same plot 'zoomed in' to better see the results for GS and PSO.

Table 9: Strength of prior, real data (Real Data = TRUE)

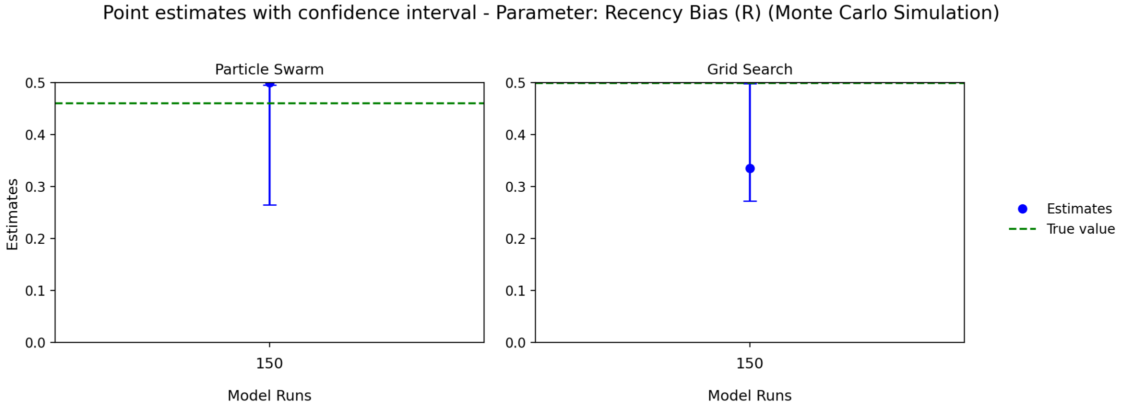
Optimization	Model Runs	Estimate	Lower CI	Upper CI
Genetic Algorithm	20	1.1381	1.0044	48.2088
Genetic Algorithm	50	1.2954	1.0047	48.6251
Genetic Algorithm	150	34.0763	1.0073	47.7466
Grid Search	20	1.2666	1.0000	1.9769
Grid Search	50	1.0000	1.0000	1.4167
Grid Search	150	1.0784	1.0000	1.2748
Particle Swarm	20	1.5662	1.0075	4.1685
Particle Swarm	50	1.4373	1.0177	3.1826
Particle Swarm	150	1.0351	1.0217	2.0112

Figure 16: Estimates on Data - Prior Strength (Z) Zoomed In



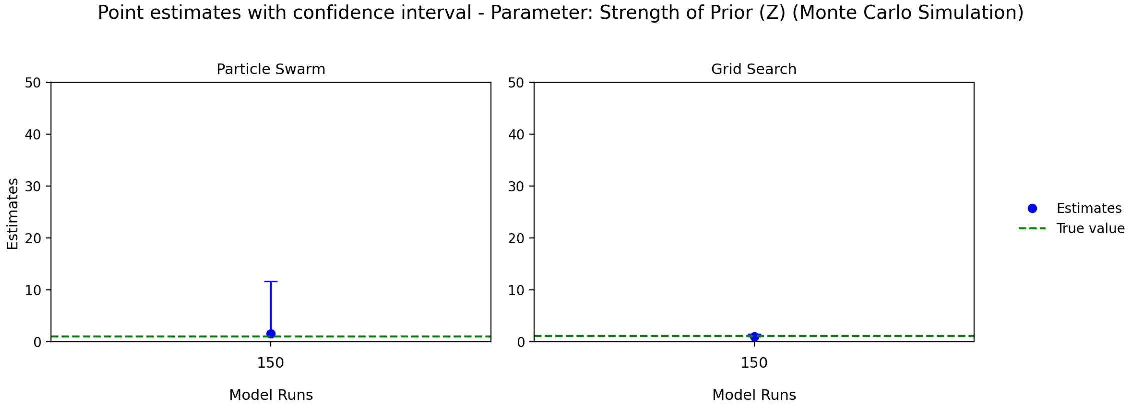
Estimate-in MCS Results: Finally, we can run MCSs for best fits and CIs again, but this time using the estimates we got out of estimating the data as the true values in our simulation. This will paint a more accurate picture of what estimates should look like if this model and estimation technique are working as intended. We report these results for both of our parameters below.

Figure 17: MCS using Estimate as True - Recency Bias (R)



Looking at our first parameter R, we first note that both true values are contained with the CIs and that CIs are of a similar size and similarly located to what falls out of our estimation of the real data. Also consistent with our MCS results from earlier, it appears GS underestimates the best-fitting R, though by a larger degree in this case.

Figure 18: MCS using Estimate as True - Prior Strength (Z)



Next, looking at our other parameter Z, we can see both plots do fairly well at finding a best-fitting parameter close to the true value. Further, both CIs contain the true value. We also note that the size of the CIs produced by GS seem consistent in size with those produced both in the past MCS and in the application to data, while the CIs produced by PS are a less precise than produced previously, though they remain reasonable.

Overall, we observe that the parameters of this model can be estimated when utilizing the right $search()$ method and hyper-parameters for the application. Further, when brought to data, our estimates both fairly reasonable and consistent with those from our MCSs where this model acts as the true data-generating process. These additional MCSs using the estimates as the true values provides a little more confidence in the validity of the method explored in this example.

Application: Information Diffusion

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.

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.

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

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:

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

where,

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

and where,

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

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).

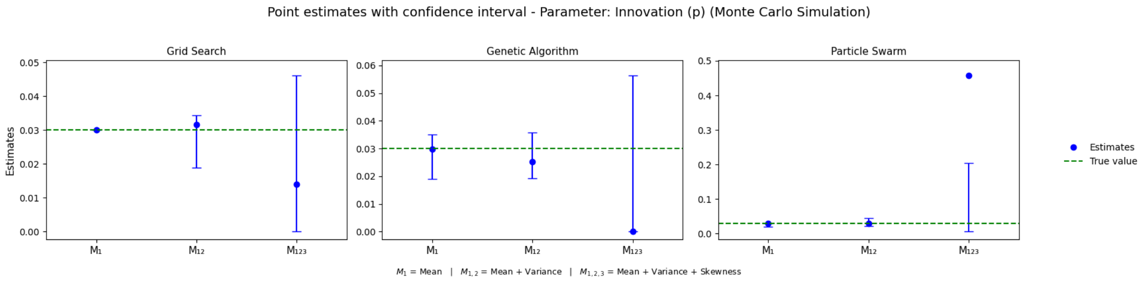
On Bootstrapping: Following the steps laid out above in Section , bootstraps can be performed by constructing a resampled dataset using our 100 blocks, then searching for parameters which best fit the data. As before, we choose to re-sample 100 times (that is, k = 100). Then, for each parameter, we drop 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 , 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.

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 19: 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

Table 10: Innovation (p), simulated data (Real Data = FALSE)

Moment	Optimization	Estimate	Lower CI	Upper CI	Diff
M_1	Grid Search	0.0301	0.0301	0.0301	0.0000
M_{12}	Grid Search	0.0317	0.0189	0.0343	0.0155
M_{123}	Grid Search	0.0141	0.0000	0.0461	0.0461
M_1	Genetic Algorithm	0.0299	0.0189	0.0351	0.0162
M_{12}	Genetic Algorithm	0.0253	0.0192	0.0358	0.0166
M_{123}	Genetic Algorithm	0.0000	0.0000	0.0563	0.0563
M_1	Particle Swarm	0.0294	0.0210	0.0304	0.0094
M_{12}	Particle Swarm	0.0290	0.0223	0.0443	0.0220
M_{123}	Particle Swarm	0.4568	0.0062	0.2045	0.1983

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 20: Monte Carlo Simulation Results - Imitation (q)

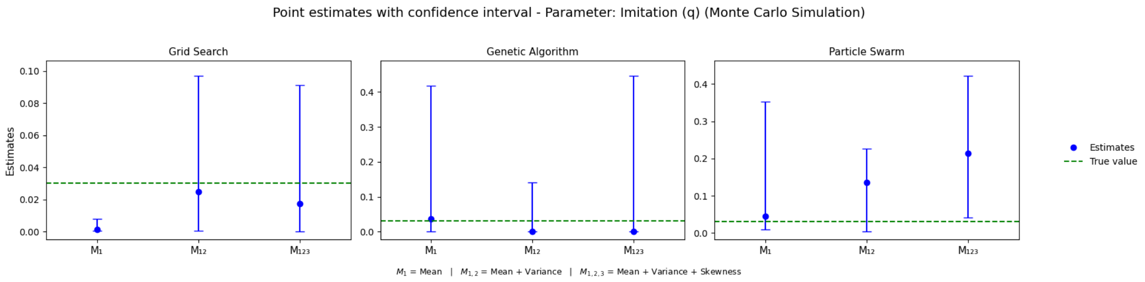


Table 11: Imitation (q), simulated data (Real Data = FALSE)

Moment	Optimization	Estimate	Lower CI	Upper CI	Diff
M_1	Grid Search	0.0014	0.0003	0.0080	0.0077
M_{12}	Grid Search	0.0246	0.0004	0.0967	0.0963
M_{123}	Grid Search	0.0176	0.0000	0.0910	0.0910
M_1	Genetic Algorithm	0.0370	0.0000	0.4168	0.4168
M_{12}	Genetic Algorithm	0.0000	0.0000	0.1409	0.1409
M_{123}	Genetic Algorithm	0.0000	0.0000	0.4451	0.4451
M_1	Particle Swarm	0.0454	0.0093	0.3530	0.3436
M_{12}	Particle Swarm	0.1348	0.0036	0.2268	0.2232
M_{123}	Particle Swarm	0.2145	0.0417	0.4210	0.3792

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 21: MCS GS Results Alternative Runs - Innovation (p)

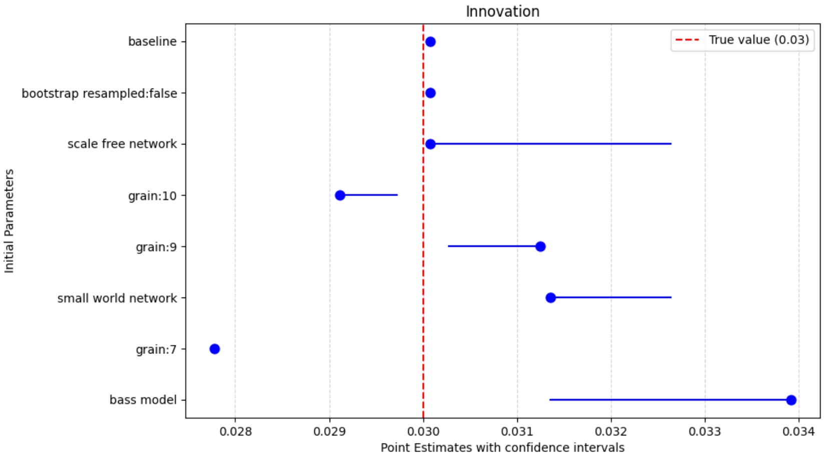
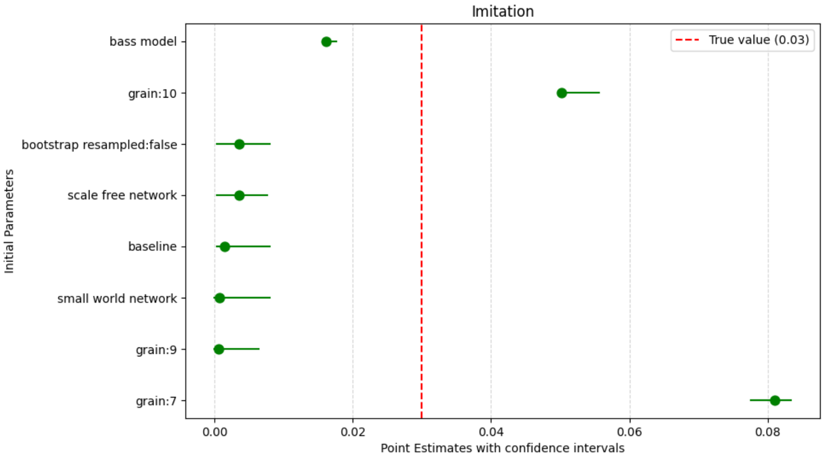


Figure 22: MCS GS Results Alternative Runs - Imitation (q)



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

Figure 23: MCS Estimated Bias Results - Innovation (p)

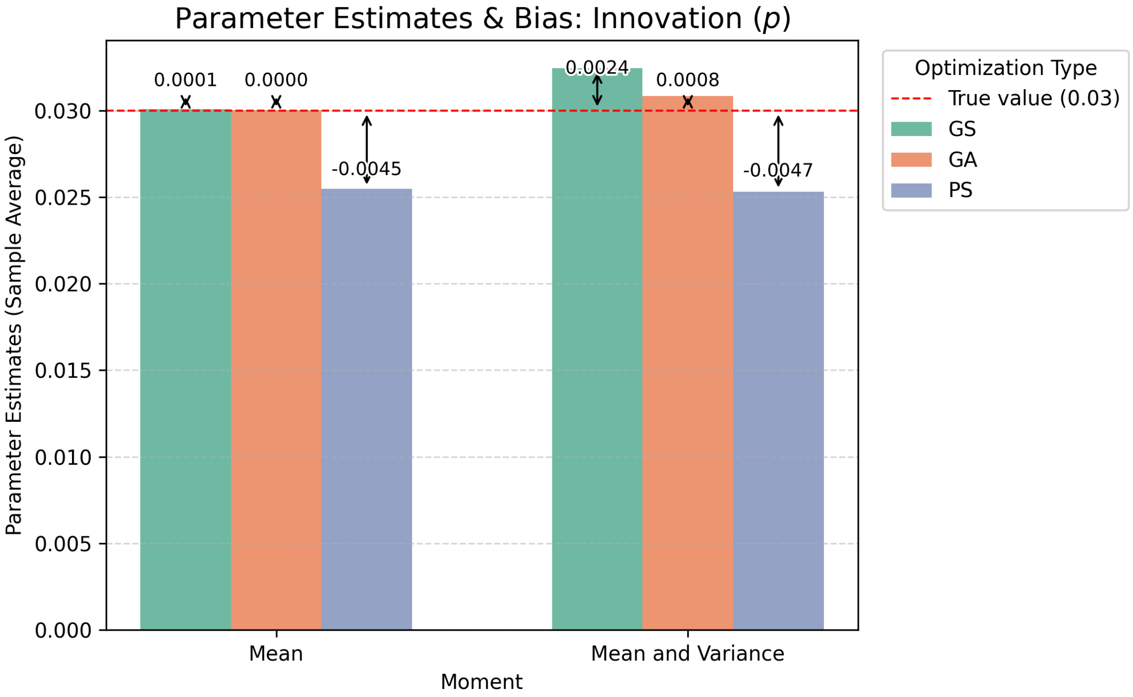


Table 12: Innovation (p): Average Estimates and Bias

Optimization	Param Ranges	Initial Param	p Average	p Bias
Mean				
GA	[0,0.5]	0.03	0.0300	0.0000
GS	[0,0.5]	0.03	0.0301	0.0001
PS	[0,0.5]	0.03	0.0255	-0.0045
Mean and Variance				
GA	[0,0.5]	0.03	0.0308	0.0008
GS	[0,0.5]	0.03	0.0324	0.0024
PS	[0,0.5]	0.03	0.0253	-0.0047

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 24: MCS Estimated Bias Results - Imitation (q)

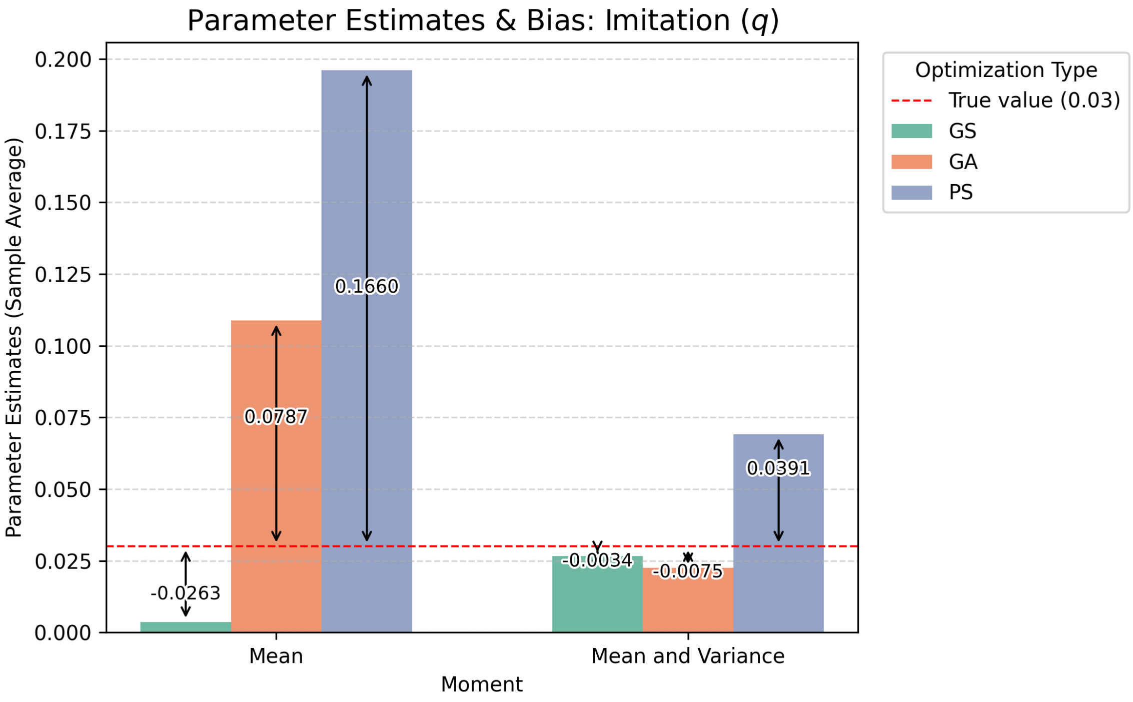


Table 13: Imitation (q): Average Estimates and Bias

Optimization	Param Ranges	Initial Param	q Average	q Bias
Mean				
GA	[0,0.5]	0.03	0.1087	0.0787
GS	[0,0.5]	0.03	0.0037	-0.0263
PS	[0,0.5]	0.03	0.1960	0.1660
Mean and Variance				
GA	[0,0.5]	0.03	0.0225	-0.0075
GS	[0,0.5]	0.03	0.0266	-0.0034
PS	[0,0.5]	0.03	0.0691	0.0391

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 25: MCS Imprecision Source Comparison - Imitation (q)

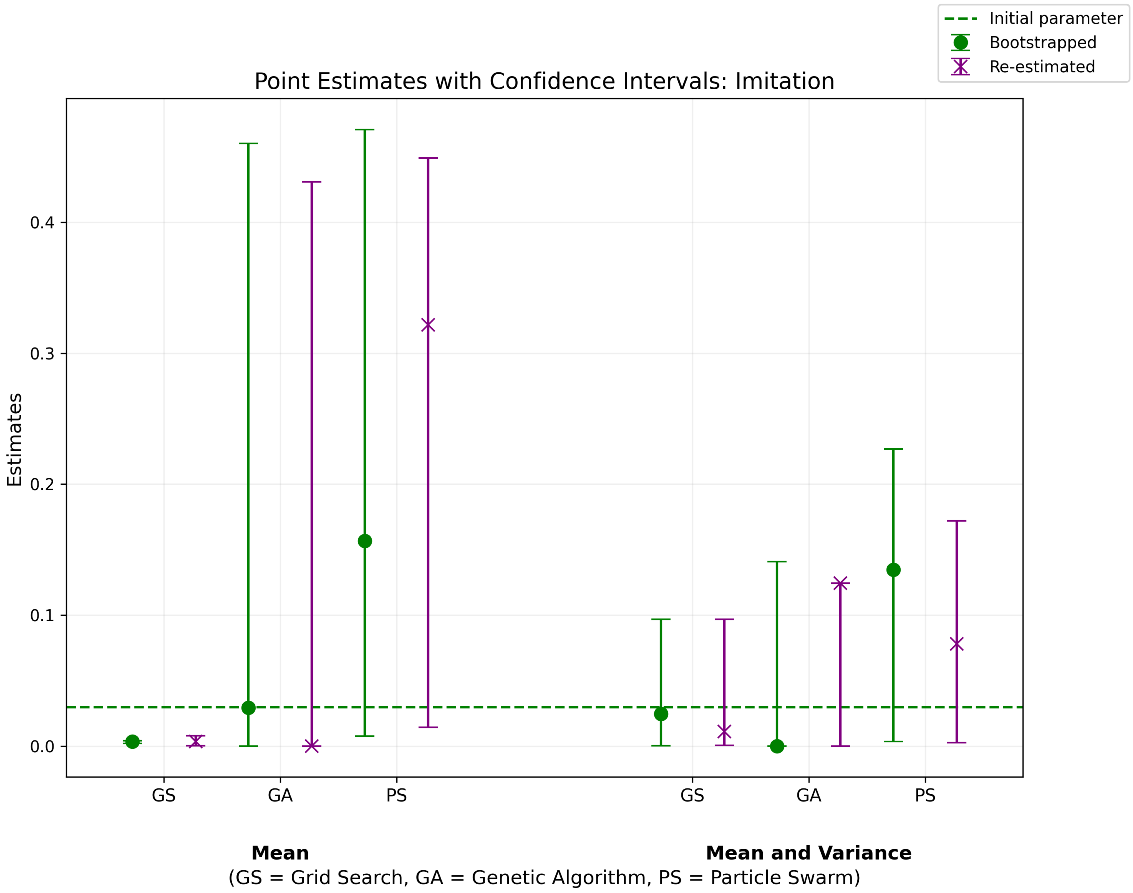


Table 14: Imitation (q): Combined Reestimates TRUE and FALSE, Sorted by Moment

Reestimates	Optimization	Moment	Best Param	Lower CI	Upper CI
True	GS	Mean	0.0014	0.0003	0.0080
	GA	Mean	0.0370	0.0000	0.4168
	PS	Mean	0.0454	0.0093	0.3530
	GS	Mean and Variance	0.0246	0.0004	0.0967
	GA	Mean and Variance	0.0000	0.0000	0.1409
	PS	Mean and Variance	0.1348	0.0036	0.2268
False	GS	Mean	0.0035	0.0003	0.0080
	GA	Mean	0.0000	0.0000	0.4308
	PS	Mean	0.3217	0.0142	0.4491
	GS	Mean and Variance	0.0110	0.0006	0.0970
	GA	Mean and Variance	0.1244	0.0000	0.1162
	PS	Mean and Variance	0.0781	0.0027	0.1721

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 26: MCS Imprecision Source Comparison - Innovation (p)

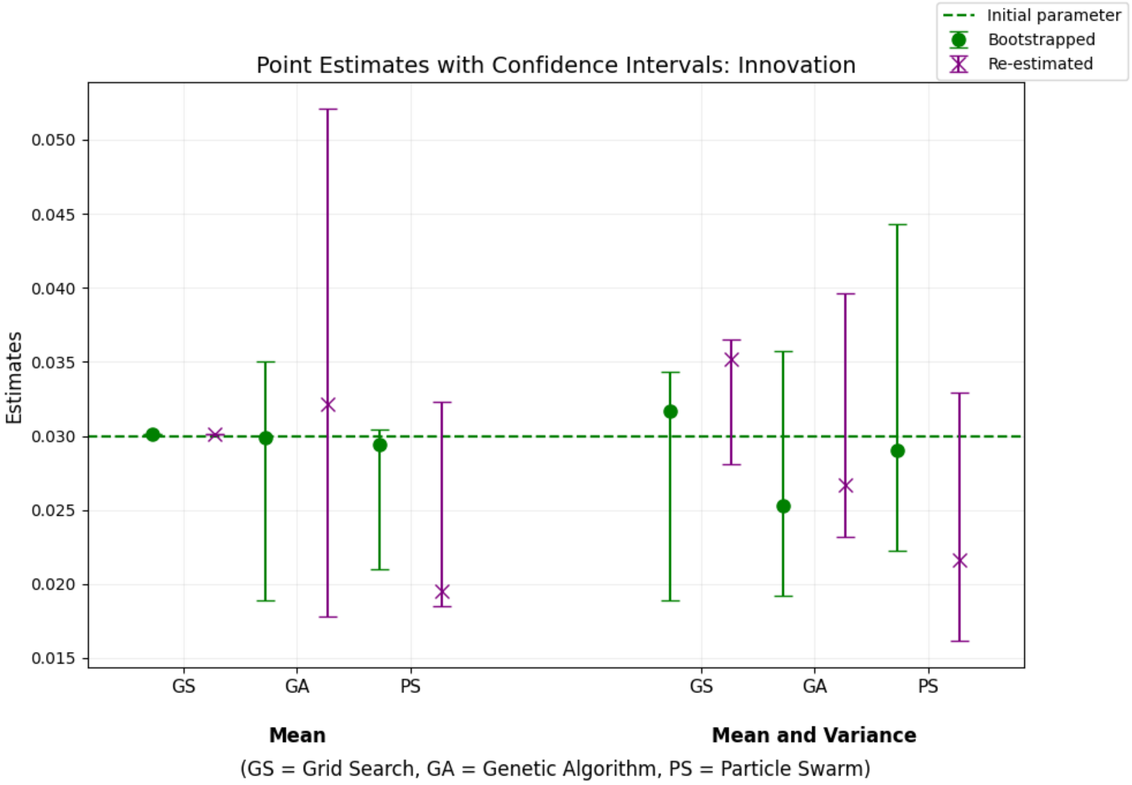


Table 15: Innovation (p): Combined Reestimates TRUE and FALSE, Sorted by Moment

Reestimates	Optimization	Moment	Best Param	Lower CI	Upper CI
True	GS	Mean	0.0301	0.0301	0.0301
	GA	Mean	0.0299	0.0189	0.0351
	PS	Mean	0.0294	0.0210	0.0304
	GS	Mean and Variance	0.0317	0.0189	0.0343
	GA	Mean and Variance	0.0253	0.0192	0.0358
	PS	Mean and Variance	0.0290	0.0223	0.0443
False	GS	Mean	0.0301	0.0301	0.0301
	GA	Mean	0.0321	0.0178	0.0521
	PS	Mean	0.0195	0.0185	0.0323
	GS	Mean and Variance	0.0352	0.0281	0.0365
	GA	Mean and Variance	0.0267	0.0232	0.0397
	PS	Mean and Variance	0.0217	0.0162	0.0329

Data Results: Taking our lessons from the MCSs we ran above, let us try to bring our model to the real data. Looking at the estimated values of p , we see fairly precise point estimates across the board. Looking to our best performers from the MCS, GA or PS with two moments in the fitness function, we observe best estimates of p are 0 or 0.0006 (which in our context is effectively 0). Note that a model with $p = 0$ will result in no transmission of information, because there will be no *innovators* to discover and share the information. This is peculiar to say the least. We also note some differences in ordering of estimate imprecision across choices of moments considered and search algorithm used is not consistent when compare to our previous MCS results. In particular, we saw previously the imprecision of p increase across the board as more moments were incorporated in the fitness function, which does not occur here when using GS or GA. Next, let us next look at q.

Figure 27: Estimates on Data - Innovation (p)

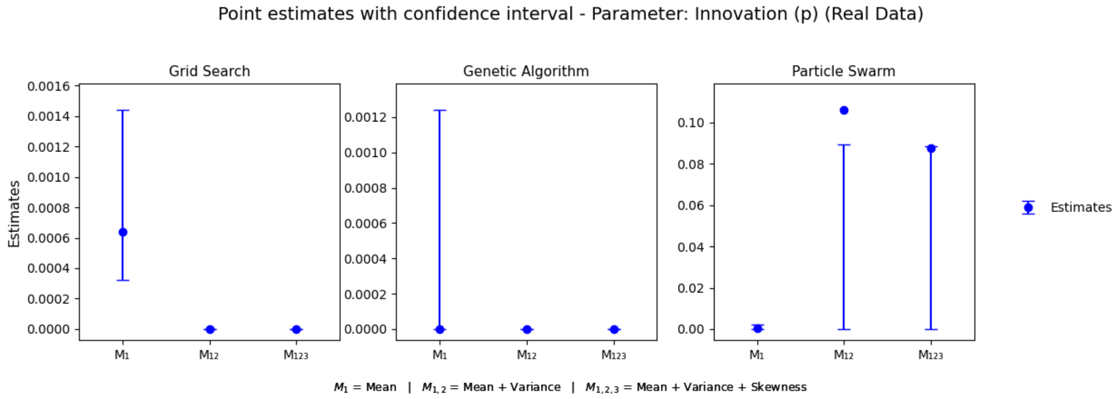
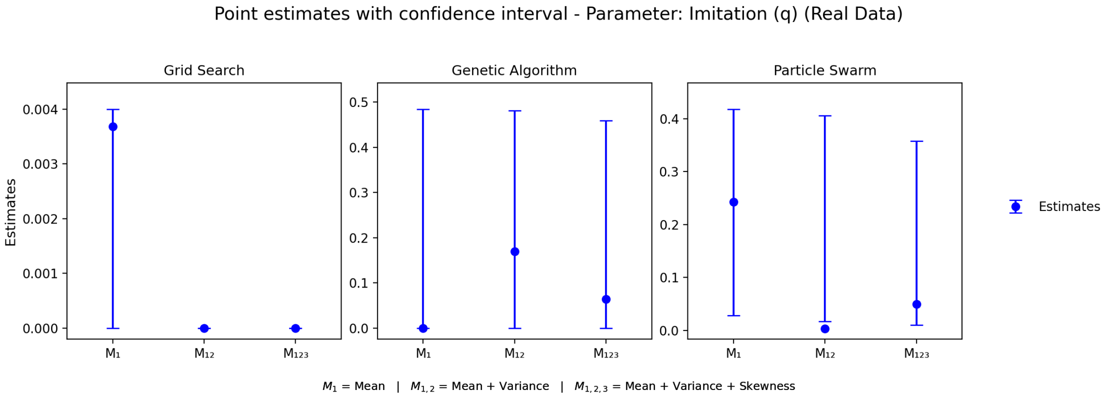


Table 16: Innovation (p), real data (Real Data = TRUE)

Moment	Optimization	Estimate	Lower CI	Upper CI	Diff
M_1	Grid Search	0.0006	0.0003	0.0014	0.0011
M_{12}	Grid Search	0.0000	0.0000	0.0000	0.0000
M_{123}	Grid Search	0.0000	0.0000	0.0000	0.0000
M_1	Genetic Algorithm	0.0000	0.0000	0.0012	0.0012
M_{12}	Genetic Algorithm	0.0000	0.0000	0.0000	0.0000
M_{123}	Genetic Algorithm	0.0000	0.0000	0.0000	0.0000
M_1	Particle Swarm	0.0006	0.0000	0.0023	0.0023
M_{12}	Particle Swarm	0.1060	0.0000	0.0893	0.0893
M_{123}	Particle Swarm	0.0874	0.0001	0.0885	0.0884

First, we recognize a broadly similar pattern in imprecision, with GS providing the much more precise estimates by and large. Looking to the point estimates of PS and GA using two moments, there is a reasonably large disagreement in best-fitting q. Across the board, q is places somewhere between 0 and 0.2, with most estimates fairly close to 0. We also do not see any noticeable improvement in estimate precision for PS or GA when using two moments instead of one or three.

Figure 28: 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, PS or GA with two moments in the fitness function). When we actually bring the model to data using these specifications, however, we see something different. First, we note that the GA estimates a best fitting p of 0, with trivial CIs $[0, 0]$, which as pointed out above, mechanically cannot be true if there is any non-zero level of information transmission. While PS recovers a more reasonable p of 0.106 with reasonable precision, the overall behavior of our estimation techniques are inconsistent with what we saw in the MCS. Since

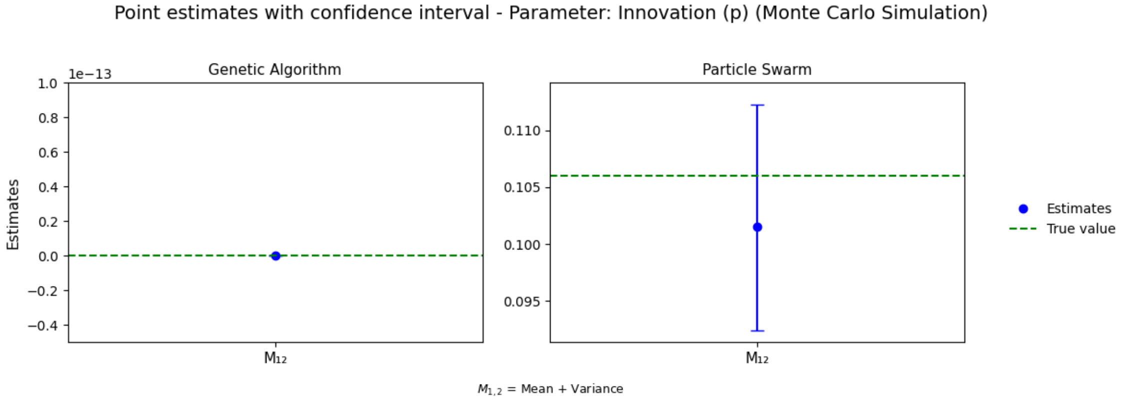
Table 17: Imitation (q), real data (Real Data = TRUE)

Moment	Optimization	Estimate	Lower CI	Upper CI	Diff
M_1	Grid Search	0.0037	0.0000	0.0040	0.0040
M_{12}	Grid Search	0.0000	0.0000	0.0000	0.0000
M_{123}	Grid Search	0.0000	0.0000	0.0000	0.0000
M_1	Genetic Algorithm	0.0000	0.0000	0.4842	0.4842
M_{12}	Genetic Algorithm	0.1700	0.0000	0.4815	0.4815
M_{123}	Genetic Algorithm	0.0644	0.0000	0.4591	0.4591
M_1	Particle Swarm	0.2430	0.0284	0.4180	0.3896
M_{12}	Particle Swarm	0.0033	0.0169	0.4056	0.3887
M_{123}	Particle Swarm	0.0499	0.0100	0.3577	0.3477

the MCS tells us about what outputs would look like if the model were precisely the underlying DGP, major inconsistencies in these results can inform that perhaps this model, with p and q assumed to be constant across cascades, is simply not valid in this context. Before taking that assessment too far, recall that our MCS was done with true values $p = 0.03$ and $q = 0.03$ which differ from the estimates returned when we applied the GA and PS to data, which were $p = 0, q = 0.17$ and $p = 0.106$, $q = 0.0033$ respectively. The accuracy and precision with which we can estimate the parameters could depend on where the true values equivalents in our model live in the parameter space.

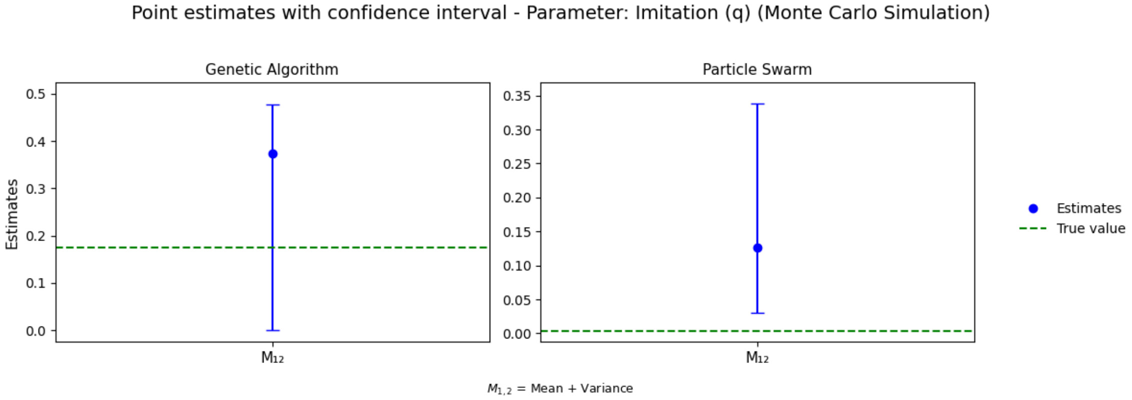
To check this, we should try our MCS again, using our estimates as the true parameters this time. We once again report these MCSs for best-fits and CIs using the estimates we got out as the true parameter values below.

Figure 29: MCS using Estimate as True - Innovation (p)



Looking at the above plots, a few things stand out. First, p and q are both relatively accurately estimated. Second, we see some similarities with levels of imprecision as we did in our estimates on data, namely that the GA has 0 width CIs while PS, estimating $p > 0$, returns non-trivial CIs, though notably they are more precise than from our data estimates.

Figure 30: MCS using Estimate as True - Imitation (q)



Now looking to q, we can see that PS misses q entirely, despite its fairly sizable CIs. The GA on the other hand does contains the true q in its confidence intervals, but by technicality since its confidence intervals seem to span the entire parameters space.

So what does all of this mean? We learned that both models seems relatively incapable of pinning down q reasonably accurately or precisely, even in our controlled MCS environment. Adding this to what we learned from the application to data, that these 'best performing models' still disagree on p by a fairly large margin, with the GA producing a best estimate for p which we know cannot be what generates the real-world information diffusion dynamics. All of this taken together, it seems reasonable to conclude that this model, with constant p and q across cascades as we additionally assumed over the baseline model, has parameters which cannot be reasonably recovered. This calls into question what our estimates on data can actually tell us about the real world.

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 a go-to method without verifying its 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

Benoît Calvez and Guillaume Hutzler. Automatic tuning of agent-based models using genetic algorithms. In Jaime S. Sichman and Luis Antunes, editors, *Multi-Agent-Based Simulation VI*, pages 41–57, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33381-4.

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.

Manuel Chica, Jose Barranquero, Tomasz Kajdanowicz, Sergio Damas, and Oscar Cordon. Multimodal optimization: An effective framework for model calibration. *Information Sciences*, 2016. doi: 10.2139/ssrn.2828069. URL https://ssrn.com/abstract=2828069. Forthcoming.

R. W. Conway, B. M. Johnson, and W. L. Maxwell. Some problems of digital systems simulation. *Management Science*, 6(1): 92–110, 1959. doi: 10.1287/mnsc.6.1.92. URL https://doi.org/10.1287/mnsc.6.1.92.

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.

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. Active nonlinear tests (ants) of complex simulation models. *Management Science*, 44(6):820–830, 1998. doi: 10.1287/mnsc.44.6.820. URL https://doi.org/10.1287/mnsc.44.6.820.

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.

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

Forrest Stonedahl and William Rand. When does simulated data match real data? comparing model calibration functions using genetic algorithms. (RHS-06-151), 2012. doi: 10.2139/ssrn.2205440. URL https://ssrn.com/abstract=2205440.

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.