A Guide and General Method for Estimating Parameters and their Confidence Intervals in Agent-Based Simulations with Stochasticity

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

February 6, 2025

Abstract

Although many Agent-Based Models (ABMs) traditionally serve to demonstrate proof-of-principle-type findings, it is becoming increasingly common and desirable for such models to be used directly for estimation. Given the increasing prevalence of using computational models for estimation in many disciplines, the need for a structured discussion on accessible and econometrically sound methods to estimate these models is of great importance.

Taking the view that ABMs are in many ways analogous to structural equation models, we detail a practical and fairly generalizable approach for bringing nearly any agent-based model to panel data in a manner akin to structural regression. We structure this paper with the aim of being an accessible guide for unfamiliar analysts to pick up and use, covering finding best fitting parameters (including summarizing and aggregating model output, establishing a fitness function, and choosing an optimization technique), estimating critical values using block-bootstrapping (including how to interpret confidence intervals and hypothesis testing in this context), and using Monte Carlo simulations to establish model/estimator properties. We also introduce a novel test to distinguish between different sources of estimate imprecision. We conclude with an example application in which we bring an ABM of learning agents playing a game to lab data to estimate agent learning parameters.

*A special thanks to the Agent-Based Policy Modeling class of 2023: Case Tatro, Drew Havlick, Illay Gabbay, Jack Phair, Jijee Bhattarai, Luke Puthumana, Muhammad Imam, Phil Siemers, Weihao Zhang, and Zhikang Tang

1 Introduction

While agent-based models (ABMs) and computational simulations may seem new, their history in economics (and in the social sciences at large) can be traced back to at least Schelling's segregation model [Schelling, 1971]. Predating the prevalence and computational power of today's computers, his model was performed using dimes and pennies 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 their 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 the subject of debate in economics.

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 for ABMs to be used directly for estimation in much the same way regressions are used. Although some great work has been done on developing elements of Agent-Based Econometric Methods (see [Bargigli, 2017]), there remains a serious lack of established and accessible 'best practices.' What is needed is a start-to-finish practitioner's guide which lays bare a methodology that is both accessible and grounded in existing econometric methods. In this paper, our aim is to do just that.

We propose a fairly generalizable methodology for bringing ABMs (and computational models more generally) to panel data with two goals in mind. First, this paper summarizes many fundamental concepts surrounding ABMs and ABM estimation, allowing it to serve as a starter guide for any interested analyst with an ABM. Second, this paper details a methodology for estimating best-fitting parameters and critical value(s), the latter of which is fairly unexplored in the context of ABMs (Guilfoos and Pape [2016]). We also discuss Monte-Carlo simulation and include a novel Monte-Carlo test which can clarify the magnitude to which different sources contribute to estimate imprecision. We conclude with an example application, demonstrating the performance of our method and highlighting the importance of Monte-Carlo simulations in establishing that our model parameters can be reasonably identified given with our chosen estimation technique.

2 Literature Review

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, in turn, 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.

In the economics simulation literature, a fair number of methodological contributions have been made. A number of publications do well to provide a general overview of some aspects of ABM estimation, [Bargigli, 2017], but leave many details of their application to the reader and make no mention of bootstrapping confidence intervals. A classic text on simulation-based methods at large [Gourieroux and Monfort, 1996] quite thoroughly details econometric considerations of computational models, but implicitly constrains much of its analysis and proposed methods to the space of models which can be fully described by a system of equations. For many ABMs, this is 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.[1] Further, this text may prove hard to engage with for some non-econometricians.

We also summarize a number of ideas informed by the structural estimation literature both within and outside economics (including Hoyle [2012] and Greene [2017]) which provide a number of useful insights, particularly on how to interpret the elements and outputs of the estimation exercise.

Finally, when formalizing the application of block-bootstrapping for critical values in our context and related Monte-Carlo simulations, a great deal of attention was given in particular to Davidson and MacKinnon [2002] and MacKinnon [2006].

3 ABMs as Structural Models

3.1 Comparing ABMs and SEMs

One way to think about bringing agent-based models (ABMs) to data is to view ABMs through the lens of structural equation models (SEMs). SEMs and ABMs

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

can both be thought of as mappings from some vector of inputs X to some vector of outputs Y which often unfolds over time given some set of parameters θ (though these mappings may not be one-to-one for some ABMs). Just like SEMs, ABMs also utilize presumed or hypothesized causal connections in these mappings which are often motivated by some combination of existing theory, empirical findings, and everyday observation. Further, in our context, ABMs are analogous to a particular type of SEM estimation technique structural regression (SR) in that, taking the structural model as given, we aim to find best fitting parameters (and test their significance) by finding parameters that minimize the loss between observed data and some moments of summarized model output over time.

While there does exist an intuitive mapping of ABM to SEM and ABM estimation to SR as given above, the use of ABMs as a SEM does require some methodological adjustment. Many of the features that serve as selling points of ABMs have also serve as unique sets of challenges during estimation. First, ABMs are extremely flexible in the types of operations they can represent by allowing for procedural / algorithmic based representations of system components in addition to typical equations. This additional flexibility can allow for the relaxation of common modeling assumptions (e.g. rational agents) in unique and possibly more realistic ways. This also means, however, that finding closed-form solutions for parameters which maximize our measure of fit is often impossible (via maximum likelihood for example). Instead, exploration of the parameter space must be done using one of a number of (often stochastic) optimization techniques which can settle on sub-optimal solutions by chance. Further, as we'll discuss later, there is a priori best way to search this parameter space. This will provide some additional challenges when estimating and interpreting our confidence intervals. Secondly, ABMs are often utilized to model non-trivial interaction between many smaller units and exhibit a degree of path dependence. This means ideally we'll need data on a non-trivial number of groups of units which may have interaction within groups, but not between groups. This will provide us with multiple, independent 'group level' observations. This is essential for creating blocks of data we can use to bootstrap for critical values. Lastly, ABMs are often path dependence and don't necessarily assume are additive. This can create some additional challenges. For a model with a stochastic component, fairly different outcomes can be produced even from the same initial conditions by chance alone. This means a number of model runs under the same conditions will need to be collected and aggregated anytime a comparison between the model to data is made. We'll cover each of these challenges and how our method aims to address them in greater detail in later sections.

3.2 Interpreting Estimates from (ABMs as) SEMs

In the following sections, we'll delve into one way to fit your model and some tests for estimate accuracy and precision. First we ought to make sense of what exactly we would learn by estimating such a model in the first place. This is precisely the purpose of this subsection. Knowing that ABMs are built upon presumed causal connections encoded using a particular functional form the modeler specifies, it can be tricky to make sense of what precisely is uncovered when we fit such a model to data. How do we interpret our parameters and confidence intervals? Can we say anything about causality?

SEMs themselves have a history of confused interpretation (see Hoyle [2012] for further discussion) but Pearl 2009 provides a resolution. Hoyle [2012] does well to summarize the inputs and outputs of SEM estimation and how to think about them. We barrow heavily from Hoyle [2012] below, and build upon this summary to clarify how this maps to the estimation exercise we'll apply to our ABM in question.

The (SEM) inference method takes three **inputs** [Hoyle, 2012]:

- A set of causal assumptions A and a model M_A that encodes these assumptions. M_A in this context is our ABM.
- A set of questions (queries) Q which the model and data can both speak to. Commonly, this takes the form 'What is the treatment effect of X on Y?'
- A set of data D which the modeler presumes is generated by a true underlying data-generating process DGP_{Data} which is consistent with the causal assumptions A.

In the case of an ABM, we argue a few **more inputs** are need to be non-trivially chosen particularly in the case of an ABM in addition to the inputs above:

- A summary function $S(.)$ which takes output from the model M_A and produces summary statistic(s) from that output which mirrors equivalent summaries of the data.
- An aggregation function $Agg(r, .)$ which takes summarized output from multiple model runs and aggregates them in a way such that a comparison to data can be made. The simplest case of this would be to get averages of your output of interest across runs, but perhaps averages of different moments are also of interest.

- A fitness function $fit(.)$ which evaluates how well the aggregated summarized model output and the summarized data satisfy your desired measure of goodness of fit.

- An optimization technique $search(\delta, .)$ which aims to return a set of parameters which maximizes your measure of fit (or minimizes loss, depending on how you characterize your fitness function). Unlike in typical SRs, maximizing fitness will often have no closed form solution, so we'll have to choose a method for searching the parameter space.

Using these inputs, the following **outputs** are produced [Hoyle, 2012]:

- A set of statements A* that are the logical implications of A. [2] These have nothing to do with data and come from the model M_A which encodes A in some way. These can be as simple statements that follow directly from A or can take the form of more complex model properties.

- A set of claims C about the magnitudes of the queries in Q which are generated using our data and are conditional on our our assumptions A (more specifically our encoding of A, M_A). These are our estimated structural parameters.

- A list of T testable statistical implications of A may also become apparent which are not utilized in the fitness function explicitly. These emergent observations can then be brought back to the data to see if they are consistent with data. This can serve as an ex-post form of partial model validation. For example, if it turns out neighboring agents have highly correlated outcomes in model output, you can determine to what degree that matches their correlations in data. This is analogous to what Wilensky and Rand [2015] call *Macro Empirical Validation*, which we'll discuss further in a later section.

In addition to these outputs, we take interest in one **more output** which is typically estimated in SRs:

- Critical values for each of the estimated structural parameters (generated with block-bootstrapping) which we can then use to establish the degree of precision with which the parameters of the model are estimated. Furthermore, we can also observe if this precision is enough to establish significance of the estimates.

[2] It may seem A* should be obvious given A, but that is often not the case. The process of modeling itself can be seen in part as a tool for formalizing and providing a method to give analysts access to A* from some set of assumptions A [Hoyle, 2012]. Along these lines, the term *Emergence*, which is commonly used in complex-systems circles, refers to properties of model output which aren't obvious from looking at the parts used to construct the model.

Importantly, our claims C about our queries Q are conditional on M_A. In words, the statement being made is "If you believe M_A, then you must also accept the claims in C." Hence, the ability to generalize the claims C to reality hinges greatly on the validity of the model in question. The inverse is also true, in that if the model has a high degree of validity, as could be the case for a meticulously validated 'digital twin' (to give an extreme case), these statement can be extremely powerful.

It is also important to make clear what establishes *validity* and what establishes if the model is *reasonably identified*. This will be made clearer in context of this method in later sections.

4 On Data

For the purposes of applying this method, we assume the modeler has panel data on hand they're interested in bringing their ABM to. This data is structured in such a way that each observation $d_{i,t}$ represents the recorded behaviors of interest by unit i at some time t, which can be broken into inputs $x_{i,t}$ and outputs $y_{i,t}$. These units i can be anything (particles, organisms, organizations, firms, countries e.t.c.) but henceforth we'll refer to these units as individuals. Presumably if we're using an ABM, we have a model in which these individuals i interact with each other at some level non-trivially. Our first task is to think about the boundaries of those interactions and where that's captured in our data.

We define a *group* g as a set of units which have no interaction with units outside the set at any point over the time period of our panel. We also denote the number of groups as G and the size of a group g as $Size_g$. Every unit should be in a group and no unit should exist in more than one group. For example, if your panel data comes from a lab experiment where groups of N players play a game over a number of rounds, a natural grouping would be the lab defined groups, of which you have N of. Groupings also arise in many natural contexts. Depending on the variables of interest, groups can be households, cities, communities, or disconnected networks. This grouping process allows us to establish the scale at which we have independent clusters of observations. Importantly, we want the groupings to contain as few units as possible without violating our interaction criteria above.

Next, we define a *block* b as all of the observations $d_{i,t}$ over time t corresponding to units i within the same group g. Formally:

$$b_g = \{d_{i,t} | \forall t, i \in g\} \tag{1}$$

These blocks will be the unit we use to resample our data when we eventually block bootstrap to generate confidence intervals. We should have a block for each group g, meaning we have G blocks. We'll denote the set of blocks $\{b_1, ..., b_G\}$ as set B. Formally:

$$B = \{b_g | \forall g \in \{1, ..., G\}\} \tag{2}$$

Note the data contained within all the blocks b_g in B is precisely the same exact data contained within D. It has simply been grouped into independent blocks.

Another assumption we make regarding the data is that it doesn't suffer from any substantial *attrition* issues. Attrition occurs when individuals drop out of the panel data over time for non-random reasons and can lead to fairly biased estimates. We deem handling such issues outside of the scope of this paper, but point readers to a number of discussions for more info, see Baltagi [2005] or [Greene, 2017].

5 Validation

5.1 About Validation

Perhaps unsurprisingly, *validity* has come to mean a number of similar but distinct things in different fields. Below we discuss what is meant by validity typically in the context of ABMs and what economists mean, why they're both valuable, and some existing methods for establishing model validity.

In [Wilensky and Rand, 2015], *Validation* (or model-to-reality validation) is defined as the process of ensuring there is a correspondence between the model in question M_A and reality. If we suppose there is some underlying Data-Generating Process (DGP) in the real world for our phenomena in question which takes some inputs X and returns some output of interest Y, then Model validation is the process of establishing reasonable similarity between this DGP, $DGP_{Reality}$ and our model M_A which we can also notate as DGP_{Model}, that takes some inputs $\hat{X}$ (which may or may not coincide with X) and returns some output.

Importantly, the model need not be a replica of reality. A good model should capture the salient features we observe in reality which we believe to be relevant to the question we aim to answer while leaving out what we might consider superfluous. These included features will often be simplified abstractions from the actual

underlying processes, but importantly should 'operate in the same spirit.' [Wilensky and Rand, 2015] go on to distinguish between a few different types of model validity which are captured in part or full in a number of other texts on ABMs and simulation (including Sayama [2015]).

First, validity of a model can be measured at multiple levels. *Micro-Validity* assesses how well the underlying components of the model match reality (e.g. agent behavior, interaction rules, etc) while *Macro-Validity* assesses how well features (including emergent features) of model output seem to match reality. Another distinction they illuminate lies in how validity is determined. What [Wilensky and Rand, 2015] call *Face validity* aims to qualitatively establish correspondence by identifying observable similarities in model features or outputs and establishing an absence of unreasonable assumptions. *Empirical validation* instead aims to establish quantitative correspondence by evaluating fitness between model generated data and real world data.

Such discussions on degree of correspondence also have a long history in economics. [Lucas, 1976] makes the case for the importance of micro-founded macro models. This contribution solidified for much of the field that a valid macro model must not only reproduce features of output, but also must do so using equations derived from interacting micro-level agents with behavior and interaction rules which are also reasonable. From this perspective, the term 'micro-foundations' is simply another name which economists have for 'micro-validity.'

While the validity concepts mentioned above may seem complete, in reality, we cannot overlook that the DGP used to create our data is often not identical to the DGP in the real world we're aiming to learn about. Hence, there are actually three types of correspondences to consider: the correspondence between DGP_{Model} and DGP_{Data}, between DGP_{Data} and $DGP_{Reality}$, and between DGP_{Model} and $DGP_{Reality}$. In economic circles, the terms *Internal Validity* and *External Validity* speak to such concerns.

Internal Validity refers to the degree to which our we can learn about the population being studied while *External Validity* refers to how much our estimates can tell us about other populations Angrist and Pischke [2008]. Internal validity can be thought of as the degree to which we can establish a correspondence between DGP_{Model} and DGP_{Data}, while external validity focuses more-so on the degree to which we can establish correspondence between DGP_{Data} and $DGP_{Reality}$. Collecting data from a lab experiment for example, can allow for a high degree of internal validity, as we have a great deal of control in both constructing and observing the circumstances under which certain outcome phenomena occur. This data is often less externally valid however, particularly in social systems, as the highly controlled

circumstances under which the data was collected may differ from the circumstances faced in the real world. Thus there is often a tension between internal validity (which establishes how well you can answer a question) and external validity (which establishes how generalizable your findings are to populations outside of the population you collected data from) when making choices about data sources.

Importantly, this notion of model-to-reality validation, in which we aim to establish how much our model can tell us about the real world, can be achieved through reasonable levels of both internal and external validity. If there is a reasonable correspondence between DGP_{Model} and DGP_{Data} and there is also a reasonable correspondence between DGP_{Data} and $DGP_{Reality}$, then there must be to some degree a correspondence between DGP_{Model} and $DGP_{Reality}$ by transitivity. Less formally, if your estimates are both internally and externally valid, then it must be the case that your model in question has some degree of 'model-to-reality' validity.

5.2 Model Validation Techniques

How to sufficiently validate a structural model is still a question subject to much debate. Below we briefly discuss several common methods employed for the purposes of model validation.

The simplest and arguably most necessary form of validation is to lay bare and make accessible your model design and results. Taking this 'Hands-Above the Table' approach with both the model design and how its output compares to the data serve as basic forms of micro and macro face validation respectively. Ultimately this clarity allows readers to perform their own qualitative assessment and, through feedback, serves as an important part of the model selection process.

Docking is another fairly common method of model validation which aims to 'barrow' the validity of existing models in a domain of study. It is common for analysts to use ABMs to extend an existing model in a discipline. This often takes the form of relaxing a number of common assumptions (e.g. well-mixing agents, rational expectations). When such an approach is taken, implementing a version of your model without assumption relaxation (which aims to emulate the discipline model you're extending) can be a powerful validation technique. If the model which the ABM extends is fairly valid, then by showing your ABM in question produces similar outcomes when you do not relax the assumption in question, the ABM is equally Micro and Macro valid to the original model under this set of assumptions. Further, the general case of the model can only be less valid to the extent that the relaxation in the model makes it so.

Another test for macro-validity is sometimes possible when unexpected patterns

in model output utilizing our best fitting parameters occurs. When this happens, a natural next step is to see if similar phenomenon appear in the real world or in data on the population in question. Importantly, this output phenomenon should not be encoded into the fitness function (i.e. we shouldn't select for it) and it should be reasonably possible for this outcome to not occur over the space of all possible parameter combinations (i.e. it's not 'cooked-in'). Intuitively the argument goes, if our model, using best fitting parameters, is producing features we see in the real world that do not have to occur and which we did not search for explicitly, then this model is demonstrating it can approximate reality pretty well.

A neighboring concept to validation, which aims to establish a reasonable correspondence between model and data is *Model Selection*, which aims to establish which model from a set of models best corresponds to some set of data. One such method which should be well defined in the context of ABMs is the lasso method Tibshirani [1996], which aims to shrink parameters to 0 by making them costly in the fitness function. This is only well defined, however, in contexts where the estimated parameters are not required to be greater than 0 for the model to be well defined.

Lastly, if multiple models are considered with fairly different functional forms, an 'out-of-sample horse race' can be considered. Simply fit each model (in the way described below in section 6) on a portion of your dataset - the training set, and then evaluate how well their output aligns with the remaining portion of your dataset - the evaluation set. Importantly, these sets should be broken up into blocks first (as discussed in section 4) before assigning them to a dataset. This method is somewhat outside of the scope of this paper, as it pertains more to model prediction performance of a phenomenon of interest, but we felt it was worth mentioning at least briefly.

6 Estimating Best Fitting Parameters

With some panel data D and an ABM M_A in hand which takes a set of parameter θ and some input data X (from D) as input and returns a vector of outputs Y_{M_A} of interest from $M_A(\theta, X)$, we'd like to find a particular set of parameters θ^* which, when used in our model M_A, produces output which emulates the salient features we see in our data D. To find θ^* we'll need to define:

- A *summary function* $S(.)$ which makes output from M_A comparable to the data D.
- An *aggregation function* $Agg(.)$ which aggregates summarized output from multiple runs of M_A.

- A *fitness function* $fit(.)$ which establishes how to compare aggregated output from M_A and D.
- An *optimization technique* $search(.)$ which searches for best fitting parameters θ^*.

With these four functions in hand, we estimate the best fitting parameters θ^* by doing the following operation:

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_D), \delta) \rightarrow \theta^* \tag{3}$$

where

$$Agg(S(M_A, \theta, X), r) \rightarrow Y_{M_A} \tag{4}$$

In other words, we're searching for the set of parameters θ^* which maximizes the measure of fit between our aggregated summarized ABM output Y_{M_A} and our summarized observations from data Y_D.

6.1 Defining a Summary Function

ABMs and their output can take many forms, so it is not uncommon for model output to not be directly comparable to the data in hand. A *Summary function* $S(M_A, \theta, X)$ can serve this need by re-forming the output from M_A into something which corresponds to what is observed in data. How a model should be summarized can vary depending on the model and the questions of interest, but often it takes the form of summary statistics (mean and variance) in some outcome achieved by a type of agents or across agents but which utilizes local information.

The Schelling model looks at individual agents moving in a positions in a lattice based on their personal preferences to be by other same-type agents. The model aims to characterize to what degree long-run, macro-level segregation can result from these simple agent-level movement decisions. Each round of the model, the model results in some configuration of agents positioned on a lattice where they have up to 4 neighbors of some type. To say something about how macro-level segregation changes over time in the model, some summary statistic of macro-level segregation needs to be collected at each time step. One easy way to do this is to assess what portion of each agent's neighbors is the same type, and then find the average portion of same type neighbors across agents. Easily we can also imagine other such summary functions, the simplest modification of which could be to capture both the average and variance in same type neighbors agents have in each configuration.

Since these models are random and path dependent, we also likely want to run the model multiple times and find the average change in time over runs. This is where our aggregation function comes in.

6.2 Defining an Aggregation Function

In a typical simple linear regression, the parameters and the error term are additively separable and the expectation of the error is 0. To get the model estimate of the expected output $E(\hat{Y})$ for a particular set of inputs X, one can simply compute what Y would be given your estimated parameters $\hat{\theta}$ and observed Xs, ignoring the error term (or rather, taking the expectation, as the errors are 0 in expectation). Since the stochastic component in many ABMs cannot be easily separated out, the same cannot be done in our context. Instead, we need to estimate our expected output computationally by running a number of runs of the model under the same conditions. Given this, a common aggregation function $Agg(.)$ simply involves taking the average of the summary statistics across runs.

$$Agg(S(M_A, \theta, X), r) = \frac{1}{r} \sum_{1}^{r} S(M_{A,r}(\theta, X)) \tag{5}$$

Note, however, that since the summary function $S(.)$ can return multiple summary statistics, $Agg(.)$ may involve taking the average for each statistic returned. For example, if you're interested in using the first two moments of output in your fitness function $fit(.)$, your summary function $S(.)$ can return both the estimated mean and the estimated variance of your run output. In such cases, the aggregation function $Agg(.)$ should find the average across the r model runs for each summary statistic of interest returned by $S(.)$ for the fitness function to use.

It may also be the case that aggregations may want to be made separately for different types of agents for example. In such cases, instead of returning a summary statistics for all agents in the model, summary statistics and aggregations can be computed separately for each type of agent.

6.3 Defining a Fitness Function

A fitness function $fit(.)$ defines a metric for evaluating how well your model is performing under the current candidate parameters θ. This serves as the objective function we'll try to optimize over. In our case, we're looking for parameters θ^*

which have our model produce the important features we observe in data. Using a familiar metaphor: if output Y_{M_A} is a completed test and Y_D is the answer key, then $fit(.)$ is the grader who takes the completed test and answer key and returns a grade. This grade is commonly referred to as *fitness* if it is something we're maximizing or or *loss* if we're minimizing. On what basis, then, should we grade our model output?

In maximum likelihood estimation (MLE), the goal is to find the parameters θ^* which, given some assumed distribution, have the highest chance to generate output Y_{M_A} that matches the output observed in data Y_D. This likelihood function can be thought of as a fitness function which MLE maximizes over. For most ABMs, unfortunately, MLE remains infeasible as a likelihood function is often not derivable, except in very specific cases.

Another common approach is to score the output using the euclidean distance between some summary statistics of model output Y_{M_A} and the output observed in data Y_D. In such cases, the distance score is the fitness function and the goal is to find the parameters θ^* which minimize that distance. Mean Average Error (MAE) and Mean Squared Error (MSE) are two such specifications of this, with MSE being far more common. If averages are taken across all agents, then the MSE minimizing fitness function can be given by:

$$fit(\theta) = \frac{1}{NT}\sum_{g=1}^{G}\sum_{t=1}^{T}(Y_{M_A,t} - Y_{D,t})^2 \tag{6}$$

where again,

$$Agg(S(M_A, \theta, X), r) \rightarrow Y_{M_A} \tag{4}$$

If separate summary statistics are computed at the group level g, then we can instead define the MSE minimizing fitness function which weights group level fitness by group size as:

$$fit(\theta) = \frac{1}{NT}\sum_{g=1}^{G}\frac{Size_g}{N}\sum_{t=1}^{T}(Y_{M_A,g,t} - Y_{D,g,t})^2 \tag{7}$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow Y_{M_A g=1}, ..., Y_{M_A g=G} \tag{8}$$

As mentioned earlier, you can also generalize this to problems where you want to match multiple moments . For example, if your aggregation function returns both a mean $Y_{M_A}^{mean}$ and a variance $Y_{M_A}^{Variance}$ of outcomes across all agents, then your fitness function could be the weighted sum of the individual MSE scores (where variance gets a weight of α) in the following way:

$$fit(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}((Y_{M_A,t}^{mean} - Y_{D,t}^{mean})^2 + \alpha(Y_{M_A,t}^{variance} - Y_{D,t}^{variance})^2) \tag{9}$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow Y_{M_A}^{mean}, Y_{M_A}^{variance} \tag{10}$$

This is something that a valid ABM may have a comparative advantage in over other methods, as simple interaction rules can capture non-trivial patterns of heteroskedasticity (changing variance over time), including convergence behaviors.

6.4 Specifying an Optimization Technique

Now that we've defined what we're looking for (a set of parameters θ^* which scores best using our $fit(.)$ function), we need to define how we're going to find it. Unlike in many simple regression models, there is rarely a closed form solution or best algorithm to find θ^*. We must instead explore the parameter space manually. This is our search algorithm $search(.)$ (a.k.a. our optimization technique).

There is an immense literature on optimization techniques for broad classes of problems, the broadest of which are referred to as *Metaheuristics*. These comprise a large number of stochastic optimization techniques which utilize some degree of randomness and pass success to strategically explore parameter spaces in search of an optimal solutions. They are particularly useful for solving difficult, nonlinear problems which have no closed form solution. This makes them ideal in many ways for applications in our context.Luke [2013] does an impeccable job at making accessible both the application and intuition of a great number of these methods, starting from the very basics.[3] Our goal is not to recreate this text within the confines of this paper, but instead to provide enough of an overview for a reader to engage with basic aspects of a few commonly used methods and the remainder of this paper.

How do these methods work? In general, these optimization methods occur over multiple rounds, during each of which *candidates* (sets of parameters) are selected and then evaluated. Additionally, nearly all methods will take some set of

[3]This text is available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/

search parameters δ to guide how the parameter space is searched (i.e. how candidate parameter sets are selected each round). The main tension in many of these optimization techniques comes down to *exploration* vs. *exploitation*. On one hand, if a set of parameters is achieving a fairly high level of fitness, it's reasonable to evaluate a candidate set of parameters which are only a slightly different, reasoning that similar inputs should produce similar outputs. This exploitation of existing well performing candidate solutions can allow for small improvements on already good solutions to be made. On the other hand, drawing parameter sets only from fairly well explored regions of the parameter space may leave other, possibly higher performing areas where the true, best fitting parameters lie completely unexplored. Therefore, a case can also be made for some level of dispersion during the search, as it will help guard against settling on local-optimum. For a taste of how these play out, we'll briefly discuss a few common methods which we later utilize in our example: *Grid-search (GS)*, the *Genetic Algorithm (GA)*, and *Particle Swarm Optimization (PSO)*. Importantly, these methods have available libraries in commonly used programming languages (e.g. Python, C++) and some are also natively supported in NetLogo as well.

A **Grid Search** is a somewhat brute-force method each round of which has three steps. First, a 'grid' of equidistant points from the parameter space is constructed. Second, each of those grid points are evaluated and the one with the highest fitness is identified. Third, the parameter space is shrunk to a smaller space around the highest fitness point. At the start of the next round of search, a new grid of points in constructed in the new smaller search space and the process continues for a number of rounds equal to the given *search depth*. While this method casts a fairly wide net of search initially, later depths do very little to guard against local-minima. This method can also be fairly computationally expensive, especially for models with many parameters. One benefit, however, is that this method of searching has no stochastic element, so using this method will result in the same estimates given the same problem.

The **Genetic Algorithm** is an interesting method of optimization which barrows from nature the ideas of natural selection and sexual reproduction and applies them to populations of fairly well performing candidate solutions (parameter sets) to create new 'generations' of candidate solutions. First, the (initially random) population of candidate solutions are evaluated. Next, some low performers are eliminated from the population and replaced by children, which are produced using pairs of high performing solutions from the population. These children get some portion of their parameters (genes) from one parent and the rest from the other (emulating genetic crossover). Then their parameters have some chance of being randomly shocked

(emulating genetic mutation). This new, resulting population is carried into the next round, and this process repeats until the search depth is reached or some convergence criteria is met. The GA is thought to be fairly robust and serves as a bread-and-butter optimization technique for all kinds of complex problems, though it is also generally computationally expensive.

Particle Swarm Optimization, like the GA, barrows ideas from nature, though this time from the flocking/swarming behavior observed by many species looking for food (e.g. birds, ants). Initially a grid of equally spaced out points is created in the parameter space and each of these points is also given an initial velocity. Each round of the search, the positions of each grid point are evaluated and the 'best solution so far' is stored. Next, each grid point moves towards the 'best solution so far with its initial velocity and some randomness added to their direction. Individual grid point velocities are also updated such that movement is slower the closer it is to the 'best solution so far' point in the parameter space. As the points move, however, note that new, better solutions can be found along the way, causing movement to the new, higher fitness location.

For more details on these and many more search methods, we highly encourage interested parties in taking a look at Luke [2013].

Which one will perform best on my problem? This is not so easily answered. There is a long literature comparing search algorithms which on various problems which aimed to uncover which method of search was superior. Then Wolpert and Macready [1997]'s No Free Lunch theorem revealed that there is no best way to search over the space of all possible problems. Ultimately, the best way to search the parameter space for a particular problem is highly dependent on features of the problem itself, specifically its *fitness landscape*. A fitness landscape is the mapping of all possible parameter combinations to the fitness that parameter set produces. If you can, imagine having a 2 dimensional parameter space (on dimensions X and Z) and plotting what the fitness of each parameter combination would be (on the Y axis). You would end up with a surface which likely has high and low points (peaks and valleys), and slopes of various degrees, hence the name fitness landscape. For those with a less vivid imagination, an example of a fitness landscape has been provided below:

[TODO: Add Fitness Landscape Pictures Here]

It is rarely, if ever, feasible to view the fitness landscape for our problem of interest, as that would require exhaustive evaluation of all possible combinations of our parameters. Therefore, features of the fitness landscape are often not obvious to the modeler a priori, so our choice of how to search the space is often limited to our intuition. For example, a more rugged landscape (one with more peaks) likely will

require more exploration as settling on local optima is much more relevant threat than if the landscape was single-peaked.

How do I know it's performing well enough then? While we can't establish what specification might be best for fitting our data, we can investigate the level of performance the specification we've chosen is achieving in a controlled environment. For any choice of $search(.)$ and δ, we can run a Monte-Carlo simulation to see how well the search method performs at returning estimated parameters which are known to us (because we chose them). This feedback will either alleviate concerns that the search method may be ill equipped to solve the problem or establish that it is, in which case we need to modify either our choice of $search(.)$ or δ until a certain level of performance is achieved. This process is discussed in a later section.

Now with $S(.)$, $Agg(.)$, $fit(.)$, and $search(.)$ established, we should have a well defined procedure for estimating best fitting parameters θ^* using the following expressions from above:

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_D), \delta) \rightarrow \theta^* \tag{3}$$

where

$$Agg(S(M_A, \theta, X), r) \rightarrow Y_{M_A} \tag{4}$$

7 Bootstrapping Confidence Intervals

So far, we've discussed finding parameters θ^* which best fit our data D. We must be mindful, however, that these estimates are not fit on data of the entire population, but rather on a sample of data D drawn from the population. This means, even if the data generating process in the real world is identical to our model M_A, and even if we're sure that θ^* is the argument that truly maximizes our fitness function $fit(.)$, θ^* may still not be very close to the true parameter values as we only have the information content in D to learn from. To better understand θ^* then, we should establish how sensitive each best fitting parameter $\theta_i^* \in \theta^*$ is to the sampling process. Put another way, we want to establish a kind of range of possible θ^*s that could result from repeating the same procedure on different samples. We argue for block-bootstrapping as one ideally suited method for establishing such ranges for an agent-based model M_A.

7.1 What is Bootstrapping?

Recall our aim is to understand how the sampling process affects our estimate θ^*. Now if we actually had many separate samples drawn from the population,

$\{D_1, ..., D_K\}$, then the problem could be somewhat trivially solved. We could quite simply fit all K of the datasets and look at the range of parameter estimates produced. We could then also establish what the inner 95% of the estimates are for each parameter to get something akin to confidence intervals for each. Bootstrapping does just this, but instead of using actual separate samples collected from the population, it constructs simulated samples $[\tilde{D}_1, ..., \tilde{D}_K]$ by resampling data with replacement from our dataset D. The argument goes that while we can't generate new samples drawn from the population directly, our sample data D was drawn from the population directly. Given that D is a representation of what we could see and how frequently we see it, we treat D as what we know about the population and draw from it instead. Since D was drawn from the population, new resampled datasets drawn from D should also be examples of dataset which could be drawn from the population. In our case, we will be using a technique known as Block-bootstrapping which takes blocks of data instead of individual observations when constructing new datasets, where a block of data is the smallest unit of data which can be considered independent from the rest of the data (as discussed above in 4).

7.2 What Can We Learn from Bootstrapping?

7.2.1 Estimate Precision

At a very basic level, confidence intervals tell us something about how precisely a parameter is estimated using the current model, fitting techniques, and data size and type. Very narrow confidence intervals on a parameter θ_i^* tells us that, to the best of our knowledge, the parameter estimate is fairly robust to variation between samples.

Note the maximum range a confidence interval can take is constrained by the range you allow your parameter search to occur over. This means one can artificially achieve fairly narrow confidence intervals by simply constraining a parameter's search range to be fairly narrow. In light of this, we recommend that in tandem with reporting confidence intervals in this way, one should also clearly report the range over which each parameter was searched.

7.2.2 Parameter Significance

In line with traditional hypothesis testing, we can also test the significance of each of our parameters. Traditionally in a hypothesis testing framework, we establish critical values which should contain some percentage of the possible estimates (often 95%) for each parameter in θ^*. We then use this range to establish whether or not a parameter is significant by observing whether this range contains 0. If 0 is

contained within this range for a two sided test or falls below the critical value for a one sided test, then we cannot confidently rule out the possibility that the parameter has no effect on our outcome variable. Hence the phrase, "We fail to reject the null hypothesis", where our null hypothesis for each parameter is that its true value is equal to 0.

Special care has to be given to interpreting results in the context of structural models, however, as a parameter may be significant *by construction*. For example, if a parameter is only allowed to be from [1,5] for your model to be well defined, then it is impossible for 0 to fall in the range of best fitting parameters regardless of the sample data drawn. Conducting a hypothesis test on this parameter will result in significance, clearly, but this should be no surprise. Agent count as a parameter is a good example of this. Simply put, a significance test is only relevant for a parameter to the extent that the parameter is allowed to not be significant.

7.2.3 Indicator of Potential Issues

Lastly, while it is certainly possible that a parameter can have a fairly large confidence interval if it is very sensitive, this can also act as a signal for a number of estimation issues. First, this may indicate that the parameter in question is just fairly sensitive. If a few parameters have fairly large confidence intervals, this could also signal *model identification* issues, as your model may have multiple parameter specifications which achieve the same output. Similarly, a fairly flexible model may *over-fit* model output, which could explain fairly large changes in parameter estimates for fairly modest changes in sample data. Finally, it may be the case that the search process used to optimize your parameter set $search(.)$ with its given hyper parameters δ or model output Y_{M_A} is fairly noisy, which is resulting in fairly different estimates. We discuss further the role noise from your model M_A and $search(.)$ can play in estimate imprecision along with a diagnostic test to measure it in a later section.

7.3 How to Bootstrap

7.3.1 1. Construct Datasets $\tilde{D}_k$

To construct our confidence intervals, we first will need to construct a number K of resampled data sets $\tilde{D}_k$ using our blocks of data. To do this, let's first recall our definition of a block from section 4. A *block* b is a set which contains all observations $d_{i,t}$ over time t corresponding to units i within the same group g. That is

$$b_g = \{d_{i,t} | \forall t, i \in g\} \tag{1}$$

with our set of all blocks defined above as

$$B = \{b_g | \forall g \in \{1, ..., G\}\} \tag{2}$$

The new dataset is constructed by simply drawing G blocks from B (which recall is just D split into independent blocks) uniform randomly with replacement. Formally

$$\tilde{D_k} = \{b^1, ...b^G | b^m \overset{\text{iid}}{\sim} U(B)\} \tag{11}$$

We draw G blocks so the new dataset has precisely the same number of blocks as the original dataset D. Note that drawing with replacement is vital, otherwise each dataset $\tilde{D_k}$ would end up identical to D. Replacement allows for grabbing some blocks multiple times and others not at all. Repeating this process K times, we can construct our set of new datasets $\{\tilde{D_1}, ..., \tilde{D_K}\}$.

7.3.2 2. Find Best Fits θ_k^* for Each

Next, we'll have to find best fitting parameters for each of these datasets. To do so, we can use our $search(.)$ method for finding best fitting parameters θ^* (given in equation 3) once on each constructed dataset $\tilde{D_k} \in \{\tilde{D_1}, ..., \tilde{D_K}\}$. We denote best fitting parameters found for a particular constructed dataset $\tilde{D_k}$ as θ_k^* . Formally,

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_{\tilde{D_k}}), \delta) \rightarrow \theta_k^* \tag{12}$$

7.3.3 3. Construct the Critical Value(s)

At this point, we should have a set of best fitting parameters θ_k^* for each resampled dataset, which we'll denote $Z = \{\theta_1^*, ...\theta_K^*\}$. We should compare this to our best fitting set of parameters θ^* which were fit on the original full sample D. For each parameter in θ^*, which we denote θ^{i*} , we find its difference with the corresponding parameter estimate in θ_k^*, denoted θ_k^{i*}, for each of the parameter sets in Z to compute errors ε_k^i. Formally,

$$\Delta^i = \{\varepsilon_1^i, ..., \varepsilon_K^i\} \tag{13}$$

where

$$\varepsilon_k^i = \theta^{i*} - \theta_k^{i*} \tag{14}$$

Importantly, we don't take absolute values of these differences, as we'll be constructing the confidence intervals using a method which does not rely on the assumption that errors are distributed symmetrically around the mean estimate.

Next, for each parameter i we construct $\Delta^i_{Ordered}$ by simply ordering Δ^i in ascending order. From this set, we can find our critical values C_i for the ith parameter in our best fitting parameter set θ^* by finding the $\frac{\alpha}{2}$th and $\frac{1-\alpha}{2}$th percentile errors in $\Delta^i_{Ordered}$ and adding them to our best fitting parameter θ^*_i. Formally

$$C_i = [\theta^*_i + \varepsilon^i_{mth}, \theta^*_i + \varepsilon^i_{nth}] \tag{15}$$

where

$$m = \lfloor K\frac{\alpha}{2} \rfloor + 1 \tag{16}$$

and

$$n = \lceil K(1 - \frac{\alpha}{2}) \rceil \tag{17}$$

$\lfloor . \rfloor$ and $\lceil . \rceil$ refer to the floor and ceiling (nearest integer below and above) respectively. These are useful to handle cases where $K\frac{\alpha}{2}$ and $K(1 - \frac{\alpha}{2})$ are not integers, and encodes the stance that the confidence interval should error on the side of being larger if need be. It is best practice, however, to choose a number of resamples for which m and n are integers.

For example, imagine choosing K = 200 and an $\alpha = 0.05$. Then the 95% of the errors computed for θ^*_i would be between the 6th and 195th entries in $\Delta^i_{Ordered}$. Further, the confidence interval for θ^*_i could be computed as $C_i = [\theta^*_i + \varepsilon^i_{6th}, \theta^*_i + \varepsilon^i_{195th}]$. Also note that while it may appear odd to add the error ε^i_{6th} for the lower bound of our confidence interval, ε^i_{6th} should be negative as we did not take the absolute values of our errors.

For a one-tailed test, a critical value can similarly be established using the following equations.

$$C_i = \theta^*_i + \varepsilon^i_{mth} \tag{18}$$

where

$$m = \lfloor K\alpha \rfloor + 1 \tag{19}$$

Finally, recall that a parameter is considered significant if we reject the null hypothesis (of $\theta_i = 0$). This occurs in a two-tailed test if $0 \notin C_i$ and in a one-tailed test when $C_i > 0$ (for non-negative parameters). For convenience henceforth, we shall refer to this set of estimated confidence intervals as C.

7.4 Runtime Concerns

Runtime is a serious concern in general when running or fitting ABMs. Perhaps the greatest detriment of this method is that it relies on many reruns of the model. Bootstrapping confidence intervals compounds this issue as the possibly expensive process of finding best fitting parameters needs to be repeated K additional times. We hope that, consistent with Moore's Law [Moore, 1965], as we see computational power continue to grow, the time required to perform this process will shrink. In the meantime, if this process proves to be too much, there are a number of options available. First, each of the K estimates (in Step 2) can be computed in parallel, which can allow you to split total runtime across multiple machines/nodes. Second, a smaller K (k=50 let's say) can also be chosen, though this affects what your effective rejection rate is.

8 Establishing Properties with Monte Carlo Simulations

The focus of this section is to investigate properties of our model and estimation strategy. Since we're 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.

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.

8.1 Monte-Carlo Simulation - What is it?

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

In our case, we know that $search(\delta, .)$ may return different results of θ^* (see equation 3), 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.

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 (see 5), 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 don't know θ or the functional form of DGP_{Data}, we can do something quite similar.

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

$$M_A(\theta, X) \rightarrow \hat{D} \tag{20}$$

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've introduced above in turn.

8.2 Test 1 - Evaluating Estimate Accuracy

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

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,

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_{\hat{D}}), \delta) \rightarrow \theta^* \tag{21}$$

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 9.2.

8.3 Test 2 - Evaluating 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 7.3, but use our simulated dataset $\hat{D}$ instead of our real data D. Formally:

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

First, we break our simulated data up into blocks which we can resample, and then using those blocks to construct k new resampled datasets.

$$\tilde{\hat{D}}_k = \{b^1, ...b^G | b^m \overset{\text{iid}}{\sim} U(B)\} \tag{22}$$

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

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

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_{\tilde{\hat{D}}_k}), \delta) \rightarrow \theta_k^* \tag{23}$$

8.3.3 3. Construct the Critical Value(s)

Finally, we use our best fitting estimates to construct critical values for each of our parameters as described in section 7.3.

$$C_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{24}$$

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).

As we'll 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.

8.4 Test 3 - Evaluating Estimate Bias

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

$$Bias^i = E(\theta^{i*}) - \theta^i \tag{25}$$

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

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

$$Bias^i_m = \frac{1}{N}\sum_{j=1}^{N} \theta^{i*}_j - \theta^i \tag{26}$$

8.5 Test 4 - 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 impreci-

sion. 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 $search(.)$ operation (which may not always find the optimum) and the aggregated model output (which may return different aggregated output each time it's 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's can be hard to know how much this second source of variation still plays a role in our confidence interval estimates.

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

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

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

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_{\hat{D}}), \delta) \rightarrow \theta_k^* \tag{27}$$

8.5.2 2. Construct the Critical Value(s)

Next, as before, we use our best fitting estimates to construct critical values for each of our parameters as described in section 7.3.

$$\hat{C}_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{28}$$

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'd expect $\hat{C}$ to be able to shrink to near zero if runs r and search parameters δ are turned up infinitely high.

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

9 An Application Example

9.1 Application Set Up

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

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

The ABM: Following directly from the experimental design, we construct an ABM M_A in which agents are initially grouped into 50 economies of size 4. Each economy plays a number of rounds corresponding precisely to their analogues in data. 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:

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:

$$Score_{i,t=0}(\hat{a}) = Z \tag{29}$$

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:

$$Prob_{i,t}(\hat{a}) = \frac{Score_{i,t}(\hat{a})}{\sum_{\forall a} Score_{i,t}(a)} \tag{30}$$

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:

$$Score_{i,t+1}(\hat{a}) = \begin{cases} (1-R) * Score_{i,t}(\hat{a}) + \pi_{i,t} & \text{if } \hat{a} = a_{i,t} \\ (1-R) * Score_{i,t}(\hat{a}) & \text{otherwise} \end{cases} \tag{31}$$

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

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

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

On Estimating Best Fitting Parameters: If the goal is to understand behavior, then the chosen *summary function* $S(.)$ 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* $Agg(.)$ to the summarized results from these model runs which simply averages the cooperation rate across runs for each period of play.

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

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.

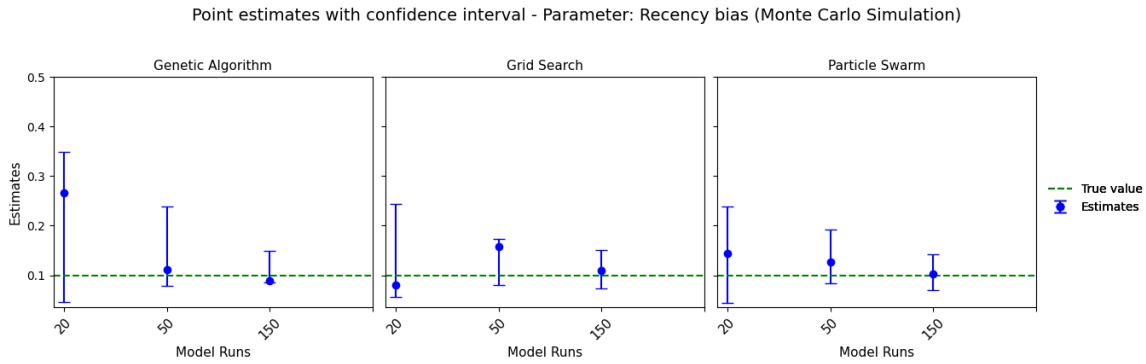
On Bootstrapping: Following the steps layed out above in section7, bootstraps can be performed by constructing a resampled dataset using our 50 blocks, then searching for parameters which best fit the data. We choose to re-sample 100 times (that is, k = 100). Then, for each parameter, we drop the outside 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 is discussed in section section8. 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.

9.2 Results

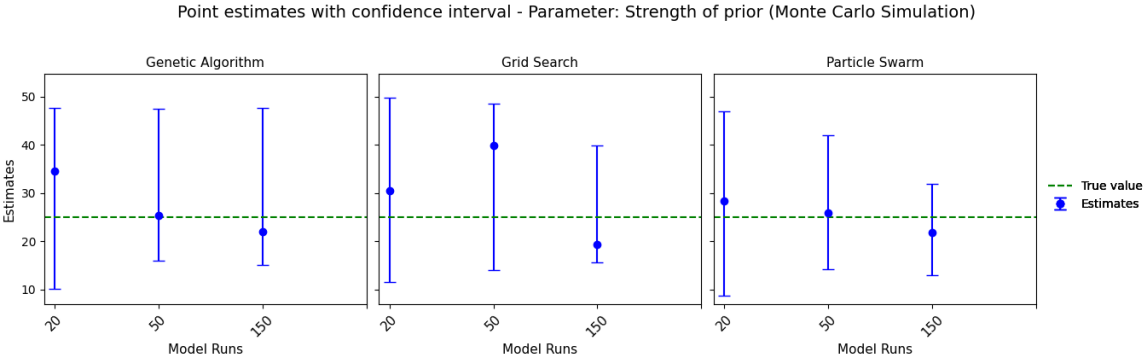
MCS Results for Best Fits and CIs: First, we show the results for both our best estimates (Test 1) and our confidence intervals (Test 2) from our Monte Carlo Simulations. We explore three levels of runs used in aggregation and three different optimization techniques, each with their own search parameters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

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



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

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



Moving to our second parameter Z, once again it seems that with the largest

number of runs, all three of the optimization techniques do fairly well at returning a best-fitting estimate close to the chosen value Z=25. It is also clear that Particle Swarm optimization 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're 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 Alternate δ Results - Recency Bias (R)

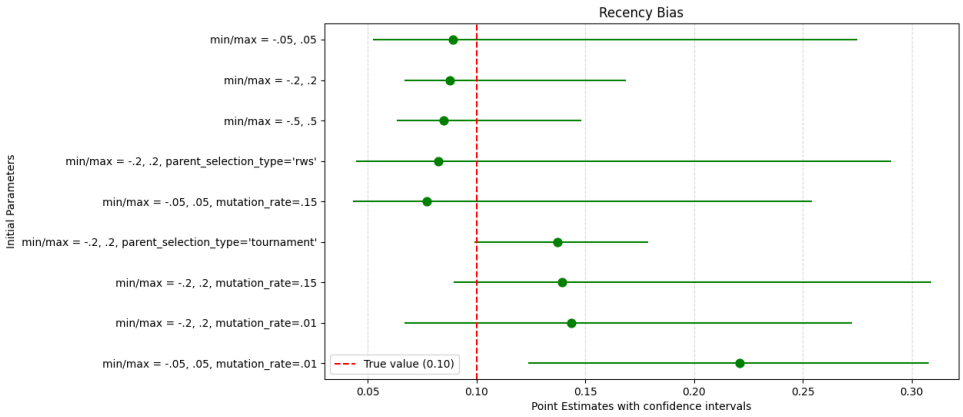
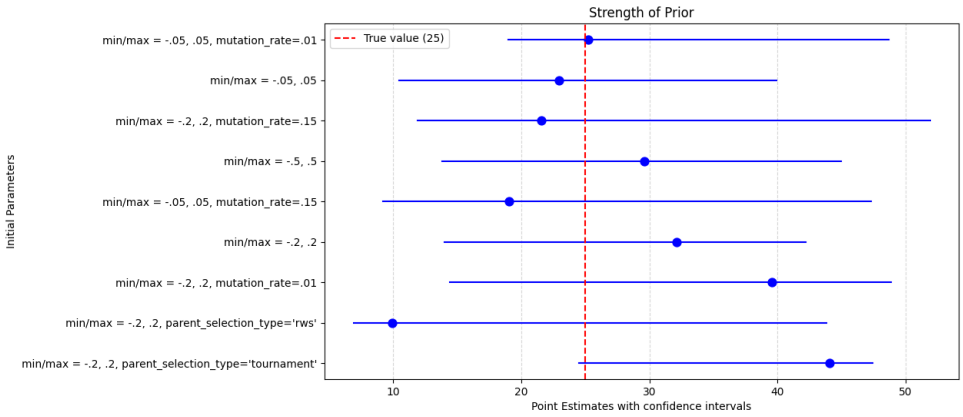


Figure 4: MCS GA Alternate δ Results - Prior Strength (Z)



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've explored above). This could also indicate that the model parameter Z cannot be reasonably identified. In our case, since we've 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 in run number. We provide two such plots below.

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

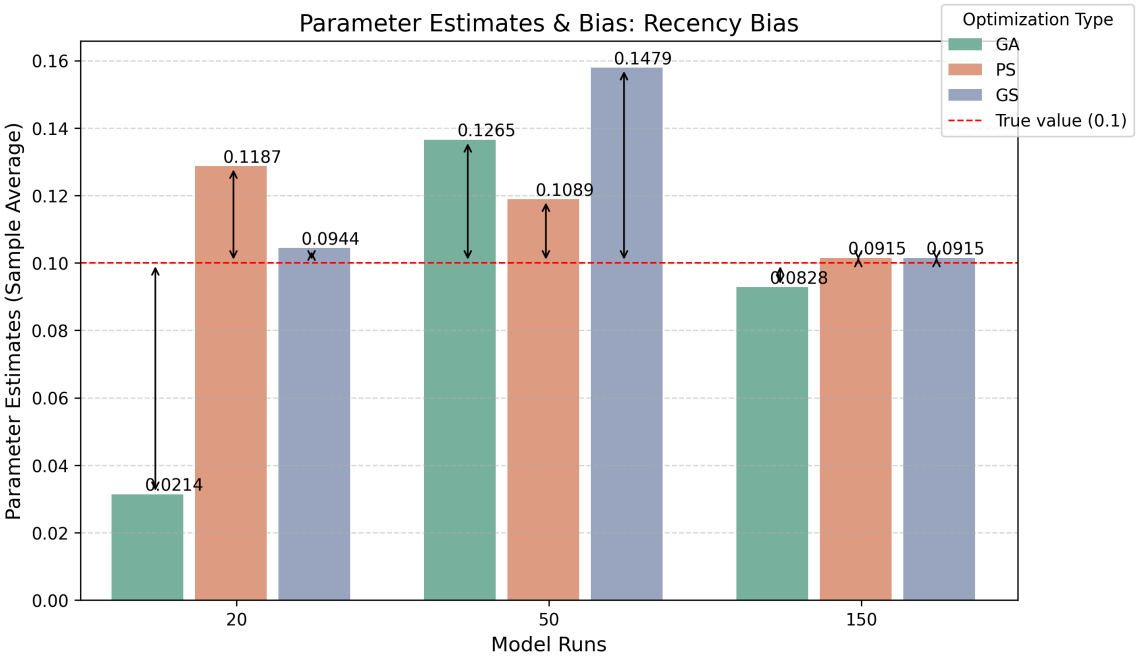
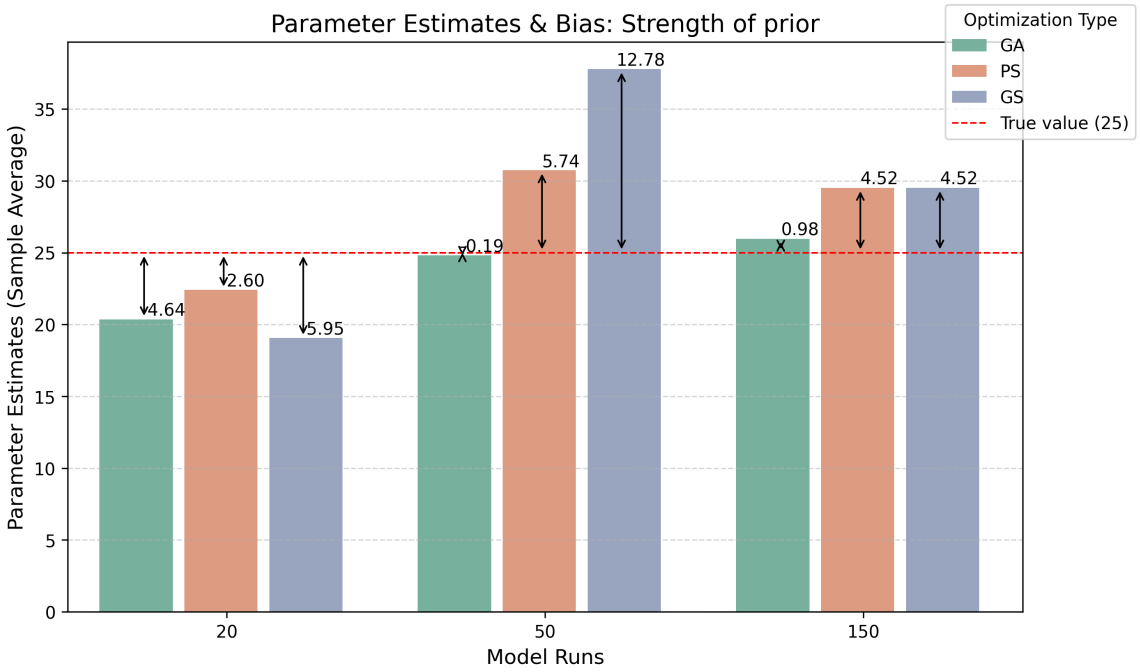


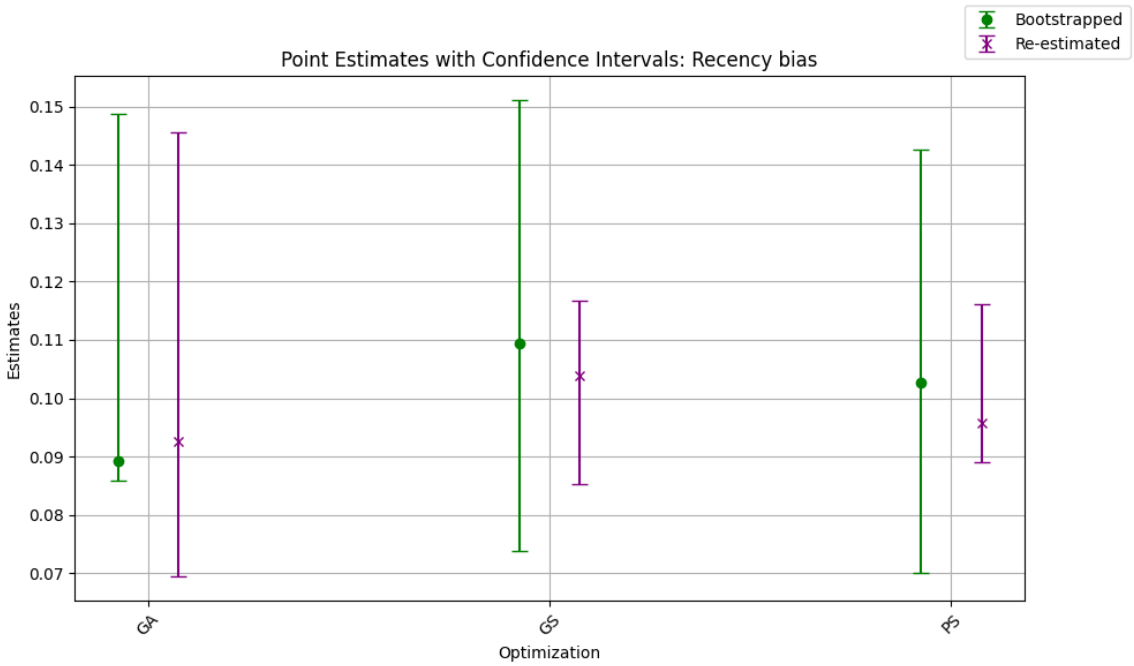
Figure 6: MCS Estimate Bias Results - Prior Strength (Z)



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

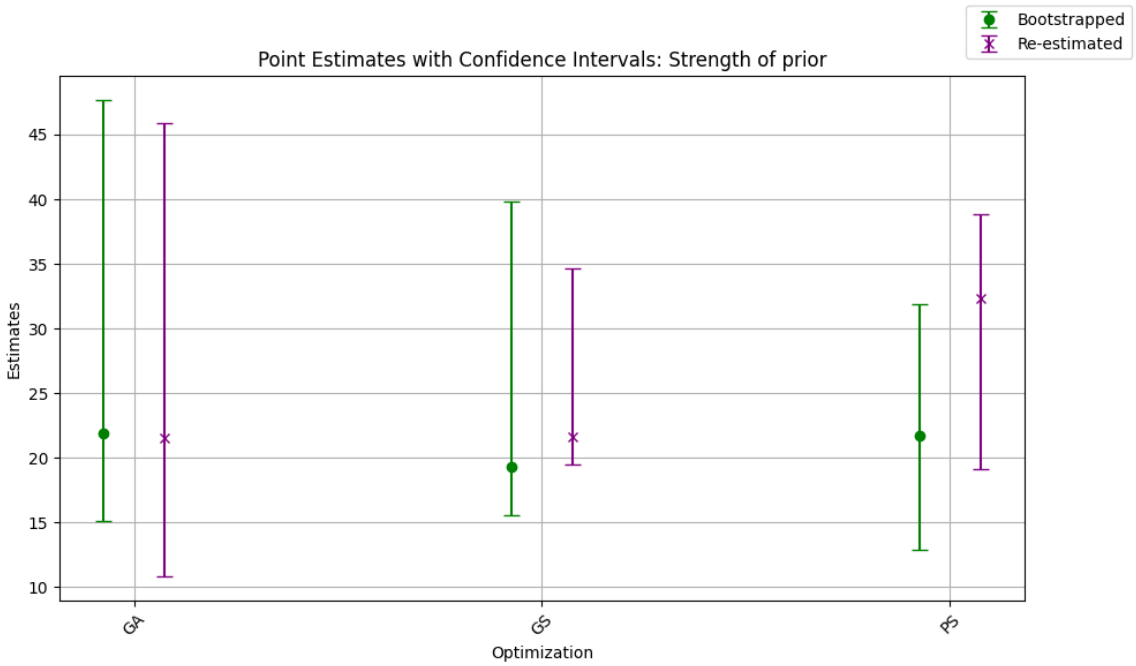
Figure 7: MCS Imprecision Comparison - Recency Bias (R)



Above, we see that for Gridsearch (GS) and Particle Swarm (PS), the role of sampling variation plays a substantial role in explaining the size of R's confidence intervals. For the GA however, it seems that the variation in the estimates of R it produces is very large even when applied on precisely the same data. This can be seen by looking at the relative size of the re estimated CI. We also see that the bootstrapped CI for R using the GA, which has an additional source of variation, is not any larger[4].

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

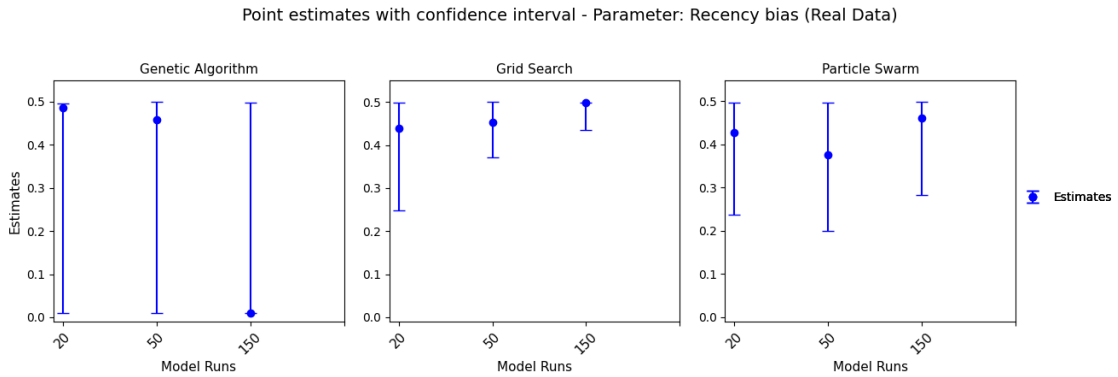
Figure 8: MCS Imprecision Comparison - Prior Strength (Z)



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.

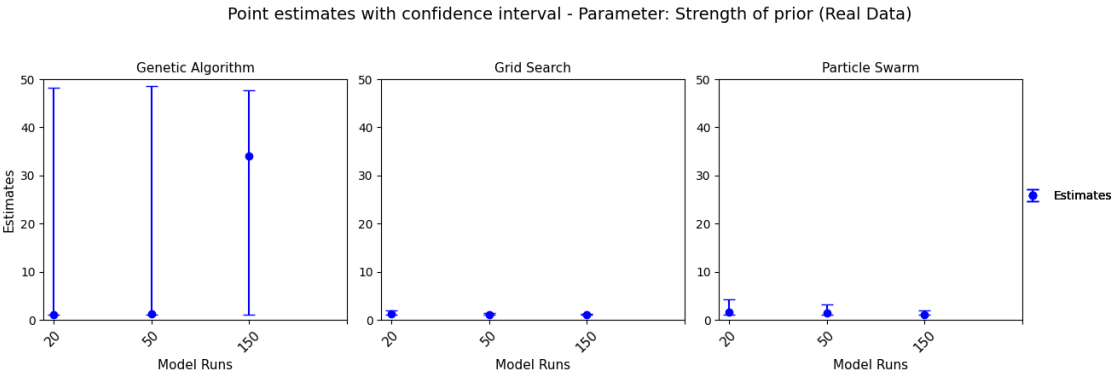
Data Results: At this point, analysts using GS or PS as their *search*(.) process with these specifications could proceed with estimation of the real data with greater confidence. Again, the process of finding best fitting parameters θ^* and CIs is precisely the same as was done in the Monte Carlo Simulations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



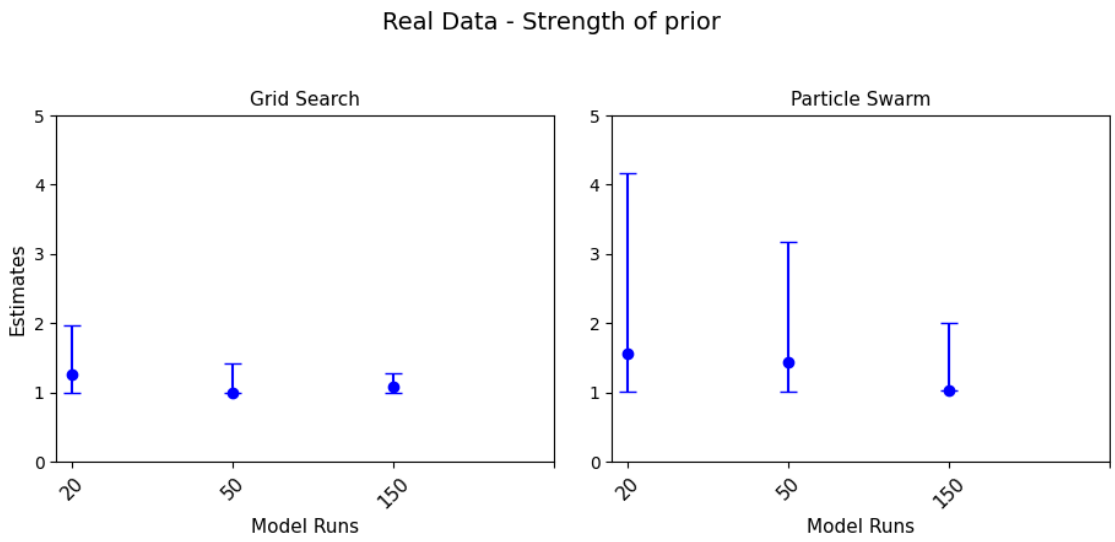
As indicated by our simulations, it seems Grid Search (GS) and Particle Swarm Optimization (PSO) at the highest level of runs (r=150) can produce much more precise estimates of R. Further, it seems both GS and PSO return similar estimates of R, with GS best estimating R = 0.50 with CI = $[0.43, 0.50]$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

Figure 10: Fitting Data - Prior Strength (Z)



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: Fitting Data - Prior Strength (Z) Zoomed In



10 Outlook and Conclusion

As the use of computational models for direct estimation becomes more prevalent in many disciplines, so does the need for grounded discussion in how to bring such models to data. Above, we've provided a practical guide for analysts to bring to a broad class of computational models to data, providing many tests and tools to establish properties about the estimation process which often go unexplored or unreported. Rather than take on faith that everything works as it should, we hope this guide can help empower analysts to demonstrate rigorously the properties of their model estimation process and to further solidify the important role that computational models can play in applied work.

References

Joshua D. Angrist and Jörn-Steffen Pischke. Mostly Harmless Econometrics: An Empiricist's Companion. Princeton University Press, 2008. ISBN 0691120358.

Badi H. Baltagi. Econometric Analysis of Panel Data. Wiley, third edition, 2005. ISBN 0470014563.

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.

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.

Todd Guilfoos and Andreas Duus Pape. Predicting human cooperation in the prisoner?s dilemma using case-based decision theory. Theory and Decision, 80(1):1–32, 2016. doi: 10.1007/s11238-015-9495-y.

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

Robert E. Lucas. Econometric policy evaluation: A critique. Carnegie-Rochester Conference Series on Public Policy, 1:19–46, 1976. ISSN 0167-2231. doi: https://doi.org/10.1016/S0167-2231(76)80003-6.

Sean Luke. Essentials of Metaheuristics. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

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.

John H. Miller and Scott E. Page. Complex Adaptive Systems: An Introduction

to Computational Models of Social Life. Princeton University Press, 2007. ISBN 9780691127026.

Gordon E. Moore. Cramming more components onto integrated circuits. Electronics, 38(8), April 1965.

Judea Pearl. Causality. Cambridge University Press, Cambridge, UK, 2 edition, 2009. ISBN 978-0-521-89560-6. doi: 10.1017/CBO9780511803161.

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.

R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society (Series B), 58:267–288, 1996.

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.

D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1):67–82, 1997. doi: 10.1109/4235.585893.