A Guide for Estimating Agent-Based Model Parameters, their Confidence Intervals, and Establishing Estimator Properties using AgentCarlo

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

Binghamton University, United States

July 7, 2025

Abstract

Although many agent-based models (ABMs) have traditionally served as tools to demonstrate proof-of-principle type findings, it is increasingly common and desirable for such models to be used directly for empirical estimation across a range of disciplines. This shift underscores the need for accessible and econometrically sound estimation methods tailored to ABMs.

This paper presents a practical and broadly applicable approach for bringing agent-based model to panel data paired with details on how such methods can be easily applied using our new Python package AgentCarlo (in development). We explain how to estimate best-fitting parameters via Simulated Method of Moments, covering the summarization and aggregation of model output, construction of a fitness function, and selection of an optimization algorithm. We also cover how to obtain critical values using block bootstrapping, including breaking data into blocks and the interpretation of confidence intervals and hypothesis testing in this context. Finally, we briefly cover the suite of Monte Carlo simulations implemented in AgentCarlo (as detailed in our companion methodological paper) which can be used to evaluate key properties of the estimation procedure.

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

1 Introduction

While agent-based models (ABMs) and computational simulations may seem new, their history in economics, and the social sciences more broadly, can be traced back at least to Schelling's segregation model [Schelling, 1971]. Predating the computational power and prevalence of today's computers, Schelling's model was implemented using dimes and pennies on a chessboard. Despite the simplicity of its rules, the model resisted closed-form analytical characterization. To understand its implications, Schelling manually performed numerous simulations from different starting conditions and then aggregated the results. With this unconventional method, he compellingly demonstrated how even a mild preference for similar neighbors can produce strikingly segregated macro-level outcomes. Since then, the role of such models in understanding emergent macro-phenomena has been the subject of significant debate.

Although many computational models, including ABMs, have traditionally been used to demonstrate proof-of-principle findings (as in Schelling's case), it has become increasingly common and desirable for such models to be used directly for empirical estimation, much like standard regression frameworks. Although important work has been done on the development of agent-based econometric methods (see [Bargigli, 2017]), there remains a serious lack of established, accessible best practices. What is needed is a start-to-finish practitioner's guide that lays out a methodology that is both grounded in econometric principles and broadly applicable paired with an easy-to-use package, which can enable low-cost utilization of these methods. This paper aims to provide exactly that. We present a generalizable approach for bringing ABMs and computational models more generally to panel data settings, and we introduce a new Python package, AgentCarlo, designed to make these methods easy to apply in practice.

First, we provide an accessible entry point for analysts of any background who seek to estimate the parameters of an ABM using the Simulated Method of Moments (SMM). We also demonstrate how this process can be implemented step-by-step with AgentCarlo.

Next, we show how simulated resampling methods can address several methodological challenges that arise when bringing complex simulations to empirical data. In particular, we illustrate how block-bootstrapping, a method familiar to many econometricians, can be used to generate confidence intervals (or more generally, critical values) for ABM parameter estimates, thereby revealing the degree of imprecision of their corresponding estimates. These critical values can, in many cases, also be used for hypothesis testing. As before, we show how this process can be carried

out with ease using AgentCarlo.

Finally, we demonstrate how AgentCarlo can be used to run a suite of Monte Carlo simulations (discussed in detail in our companion paper [**?**]). Such simulation methods allow researchers to assess, among other things, whether the model parameters of interest are reasonably identifiable.

2 Literature Review

How best to bring ABMs to data in a broad sense has remained in many ways illusive and fairly non-standardized across disciplines. For example, there is a small but growing number of texts on Agent-Based Modeling and computational modeling at large (Sayama [2015], Wilensky and Rand [2015]) and in the context of social systems more specifically (Miller and Page [2007], Tesfatsion and Judd [2006], Schmedders and Judd [2014], Romanowska et al. [2021]). While each of these texts provides an in-depth analysis of many important features of ABMs, including laying out design principles and exploring important past or potential future applications, none provides a thorough treatment of how one should bring such models to data or how to interpret the meaning of such estimates.

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

Because of this, many economists utilizing computational models see Simulated Method of Moments (SMM), a fairly generalizable method for estimating parameters, as their go-to estimation technique of choice. SMM was first introduced in McFadden [1989] and is summarized in a number of econometric texts, including a classic text on simulation-based methods [Gourieroux and Monfort, 1996]. Although these texts provide thorough coverage of some aspects of SMM, they may prove challenging for non-econometricians to engage with and are not adapted to address some of the unique challenges that arise when trying to estimate ABMs in particular. Therefore,

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

we feel part of our value added is to lay bare this method for parameter estimation and to discuss its application and interpretation in the context of ABMs. This includes providing a number of useful insights, particularly on how to interpret the elements and outputs of ABM estimation, many of which are summaries or extensions of a number of ideas from the existing structural estimation literature within and outside economics (including Hoyle [2012] and Greene [2017]).

We also formalize the application of block-bootstrapping for critical values in the estimated parameters, which we argue is a tool ideally suited for such contexts. When generating these discussions, a great deal of attention was given in particular to Davidson and MacKinnon [2002] and MacKinnon [2006].

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 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). 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 from ABM to SEM and ABM estimation to SR as given above, the use of an ABM as a type of SEM does require some methodological adjustment. Many of the features that serve as selling points of ABMs can also present unique challenges during estimation. First, ABMs are extremely flexible in the types of operations they can represent, since they allow 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 may settle on sub-optimal solutions by chance. Further, there is no 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 interactions between many smaller units and exhibit a degree of path dependence. This means ideally we want 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 which we can then use to create blocks of data to bootstrap for critical values without having to make certain undesirable assumptions.

Lastly, ABMs are often path dependent and do not necessarily assume errors are additive. This can create some additional challenges. For a model with a stochastic component, it is possible for fairly different outcomes to 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 and data is made. We will 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 will dive into one way estimate best-fitting parameters and critical values, but 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 or what is learned if a hypothesis test is conducted using the estimated critical values. How do we interpret our parameters and confidence intervals? Can we say anything about causality?

SEMs themselves have a history of confused interpretation[2], but Pearl 2009 provides a resolution. Hoyle [2012] does well to summarize these ideas as inputs and outputs of SEM estimation and how to think about them. We borrow heavily from this summary, mirroring and building upon this summary to clarify how this maps to ABM applications.

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

- A set of causal assumptions A and a model M_A that encodes these assumptions.

[2]For further discussion, see Hoyle [2012]

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** 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(s) Y_{M_A} from the model M_A and produces summary statistic(s) from that output. These summary statistics are often moments of one or more outputs of interest and will be compared to equivalent summaries of some variable(s) in 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. We will follow the paradigm of simulated method of moments.

- 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 structural regression, maximizing fitness will often have no closed form solution, so we will 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. [3] These come from the model M_A which encodes A in some way. While such phenomenon

[3] 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 are not obvious from looking at the parts used to construct the model.

may also appear in data, it is important to note that A* has nothing to data with the data in particular. 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 will 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.

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 statements can be extremely powerful.

4 Validation

4.1 About Validation

In the previous section, we discussed the importance of *validity*, but what precisely does validity mean? Perhaps unsurprisingly, *validity* has come to mean a number of similar but distinct things in different fields. Below we discuss what is typically meant by validity in the context of ABMs, what economists mean when they talk

about validity, and how these concepts overlap. We talk about why they are both valuable and some existing thoughts on and 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 in 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 the data from the real world.

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 what [Wilensky and Rand, 2015] call '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 are 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 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 or because the participants for the study may not be completely representative of the population we are interested in learning about. Thus there is often a tension between internal validity (which establishes how well you can answer a question for the sample you have) 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 reasonably internally and externally valid, then it must be the case that your model in question has some degree of 'model-to-reality' validity.

4.2 Model Validation Techniques

How to sufficiently validate structural models (including ABMs) is still a question subject to much debate. We briefly discuss below several common methods employed for the purposes of model validation.

Docking is a common method of model validation which aims to borrow 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 which is being extended) 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 where assumptions are relaxed can only be less valid to the extent that the way the relaxation is implemented 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 should not 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 is 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*. The goal of model selection is to establish which model from a set of models best corresponds to the underlying DGP used to create the data. One such method which can sometimes 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 method may not be possible, however, in contexts where some estimated parameters are required to be non-zero 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 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 5) 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.

Finally, an argument can be made that the simplest and most necessary way to

demonstrate reasonable model validity to readers 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 allows readers to perform their own qualitative assessment and, through feedback and extension, serves as an important part of the model selection process.

5 On Data

For the purposes of applying the method presented in this paper, we assume the modeler has panel data on hand they are 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 etc.) but henceforth we will refer to these units as individuals. Presumably if we are 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 units 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 does not 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].

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 next want 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 will need to define:

- A *summary function* $S(.)$ which produces summary statistics of output Y_{M_A} from the model M_A (using some set of parameters θ) comparable to summary statistics of data. In our case, these summary statistics will be the first n moments $\mu_1^{'}, ...\mu_n^{'}$ of each of the outputs of interest. We denote these summarized outputs from the model and data $Z(Y_{M_A}, \theta)$ and $Z(Y_D)$ respectively.

- An *aggregation function* $Agg(.)$ which aggregates summarized output across r runs of M_A. We denote this aggregated set of model outputs $\bar{Z}(Y_{M_A}, \theta, r)$

- A *fitness function* $fit(.)$ which establishes how to compare aggregated output from M_A, $\bar{Z}(Y_{M_A}, \theta, r)$ against the summarized output from data D, $Z(Y_D)$.

- An *optimization technique* $search(.)$ which searches for parameters θ^* which maximizes our fitness function $fit()$ using some search parameters δ.

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

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

where

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

In other words, we are searching for the set of parameters θ^* which maximize the measure of fit between our aggregated summarized ABM output and our summarized observations from data.

6.1 Defining a Summary Function

A *Summary function* $S(M_A, \theta, X)$ serves two purposes. The first purpose is to quite literally define what our outputs of interest Y_{M_A} are from the model M_A and to do any processing required to get those outputs. This step may be trivial in many cases, but not always. These outputs of interest from $S(M_A, \theta, X)$ should be comparable to the outputs of interest in our dataset Y_D. Once the model outputs of interest are decided and formalized, the second function of $S(M_A, \theta, X)$ is to use these outcomes to produce summary statistic(s) which will be the final output of this function. Since we focus on the use of simulated method of moments (SMM), these outputs will be the first n moments of Y_{M_A}, $\mu_1^{'}, ...\mu_n^{'}$, of outputs for each period of the model for which you have panel data. One or two moments are common (that is, using the mean or mean and variance of output each period), but higher moments may also be desirable depending on the problem. Let us take a look at a quick example of an ABM and how one might summarize its output.

In the Schelling model, two types of agents make individual decisions about whether to move to a new position in the lattice or not based on their preference for being around other agents of the same type. The model aims to characterize to what degree long-run, macro-level segregation can result from simple agent-level movement decisions based on small biases. Each round of the model results in some configuration of agents positioned on the lattice, where each agent can be adjacent to up to four neighbors of some type. To say something about how macro-level segregation changes over time in the model, the modeler must decide what constitutes individual level segregation (i.e. what defines Y_D), and then must aggregate across those individual levels to form the summary statistic of macro-level segregation 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 (the first moment) of same type neighbors across agents each period $\mu_{1,t=0}, ...\mu_{1,t=T}$. So the outputs in Y_{M_A} in the portion of same type neighbors each agent has each period, and the summarized output would be the first moment of (i.e. the average across) these agent-level

outputs each period. We can also imagine another such summary function which captures both the average and variance of same-type neighbors agents have each period $\mu_{1,t=0}, ...\mu_{1,t=T}$, $\mu_{2,t=0}, ...\mu_{2,t=T}$. If there are a different number of agent types, another possible summary function of interest may calculate these moments for each type separately $\mu_{1,type=A,t=0}, ...\mu_{1,type=A,t=T}$, $\mu_{1,type=B,t=0}, ...\mu_{1,type=B,t=T}$. As listing out these moments can be cumbersome, we denote this set of summary statistics $Z(Y_{M_A}, \theta)$. Note a similar set of summarized variables (moments) should also be computed for Y_D, $Z(Y_D)$, which will be used later when fitting our model.

Since these models are random and may be path dependent, we also want to run the model multiple times to approximate the expected (average) change in these moments over time. 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, and since ABMs can allow for multiple equilibria, often the same cannot be done in our context. Miller and Page [2007] refer to this phenomenon as "comparing clouds to clouds." One way to address this is 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 $S(.)$ returns multiple summary statistics, $Agg(.)$ will involve taking the average for each statistic returned separately. For example, if you are interested in using the first two moments of an outcome variable, your summary function $S(.)$ will return both the estimated mean and the estimated variance of your run output for each period of your run. In such cases, the aggregation function $Agg(.)$ should find the average across the r model runs for each summary of the two moments for each of the T+1 periods in the simulation.

The returned set of expected moments is denoted $\bar{Z}(Y_{M_A}, \theta, r)$ and we denote each moment in this set $\bar{\mu}_{M_A,t}$.

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 will try to optimize over. In our case, the important features we are looking to closely produce with parameters θ^* are the moments of output of interest from our data $Z(Y_D)$. Using a familiar metaphor: if summarized output $\bar{Z}(Y_{M_A}, \theta, r)$ is a completed test and $Z(Y_D)$ is the answer key, then $fit(.)$ is the grader that takes the completed test and answer key and returns a grade. This grade is commonly referred to as *fitness* when it is something we are maximizing and *loss* when our goal is to minimize this score. 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}, $\bar{Z}(Y_{M_A}, \theta, r)$ and the summary statistics of output observed in data Y_D, $Z(Y_D)$. In such cases, the distance score is the fitness function and the goal is to find the set of 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.

In the case of the Simulated Method of Moments (SMM), the summary statistics in question which we are finding the distance between are moments of each output of interest. If only the first moment is taken for one output variable of interest (that is, if our set of expected moments $\bar{Z}(Y_{M_A}, \theta, r)$ only contains moments $\bar{\mu}_{M_A,1,t=0}, ..., \bar{\mu}_{M_A,1,t=T}$) , then the MSE minimizing fitness function can be given by:

$$fit(\theta) = \frac{1}{T+1} \sum_{t=0}^{T} (\bar{\mu}_{M_A,t} - \bar{\mu}_{D,t})^2 \tag{6}$$

where,

$$Agg(S(M_A, \theta, X), r) \rightarrow \bar{Z}(Y_{M_A}, \theta, r) = \{\bar{\mu}_{M_A,t=0}, ..., \bar{\mu}_{M_A,t=T}\} \tag{7}$$

If multiple moments are taken, or if there are multiple output variables of interest derived from the behavior or outcomes of all agents, then our MSE minimizing fitness function can be adjusted to find the MSE of each output variable separately, then add these scores together with some weight α on the second. For two variables or moments, this can look like the following:

$$fit(\theta) = \frac{1}{T+1}\sum_{t=0}^{T}[(\bar{\mu}_{M_A,1,t} - \bar{\mu}_{D,1,t})^2 + \alpha(\bar{\mu}_{M_A,2,t} - \bar{\mu}_{D,2,t})^2] \tag{8}$$

where,

$$Agg(S(M_A,\theta,X),r) \rightarrow \bar{Z}(Y_{M_A},\theta,r) = \{\bar{\mu}_{M_A,1,t=0},...,\bar{\mu}_{M_A,1,t=T},\bar{\mu}_{M_A,2,t=0},...,\bar{\mu}_{M_A,2,t=T}\}, \tag{9}$$

α determines the relative importance the second variable has in the overall fitness score and is decided by the modeler a priori. If these are moments, it is not uncommon to give higher moments less importance in the fitness function by setting $\alpha < 1$. The assignment of a weight $\alpha \neq 1$ can also be useful when the variables of interest are on different scales, even when working with just the first moment of output. In particular, it is very common to normalize each squared difference, $(\bar{\mu}_{M_A,t} - \bar{\mu}_{D,t})^2$, by dividing by the associated moment in data $\bar{\mu}_{D,t}$. Such a specification puts the size each difference in moment contributes to the overall loss on the same scale. Formally, this looks like:

$$fit(\theta) = \frac{1}{T+1}\sum_{t=0}^{T}\frac{1}{\bar{\mu}_{D,t}}(\bar{\mu}_{M_A,t} - \bar{\mu}_{D,t})^2 \tag{10}$$

where,

$$Agg(S(M_A,\theta,X),r) \rightarrow \bar{Z}(Y_{M_A},\theta,r) = \{\bar{\mu}_{M_A,t=0},...,\bar{\mu}_{M_A,t=T}\} \tag{7}$$

If your output variables of interest are agent-type specific, then the influence of these output variables on our fitness should be scaled to account for the relative size of data points associated with each type. For example, if your aggregation function returns just the first moment for a variable of interest for agents of type J and type K separately (ie $\bar{\mu}_{M_A,type=J,t}$ and $\bar{\mu}_{M_A,type=K,t}$ for each t), where there are $Size_J$ agents of type A and $Size_K$ agents of type B with N agents total (so $Size_J + Size_K = N$) the MSE minimizing loss function could be described as follows:

$$fit(\theta) = \frac{1}{T+1}\sum_{t=0}^{T}\Big[\frac{Size_J}{N}(\bar{\mu}_{M_A,t,\text{type}=J} - \bar{\mu}_{D,t,\text{type}=J})^2 + \frac{Size_K}{N}(\bar{\mu}_{M_A,t,\text{type}=K} - \bar{\mu}_{D,t,\text{type}=K})^2\Big] \tag{11}$$

where,

$$Agg(S(M_A,\theta,X),r) \rightarrow \bar{Z}(Y_{M_A},\theta,r) = \{\bar{\mu}_{M_A,t=0,\text{type}=J},\ldots,\bar{\mu}_{M_A,t=T,\text{type}=J}, \bar{\mu}_{M_A,t=0,\text{type}=K},\ldots,\bar{\mu}_{M_A,t=T,\text{type}=K}\} \tag{12}$$

Note that fitting higher moments or different agent outcomes are often things we suspect a valid ABM may have a comparative advantage in over other methods, as the inclusion of simple interaction rules which underlie many ABMs are capable of capturing a wide array of fairly distinct patterns of behavior, including non-convergent behavior.

6.4 Specifying an Optimization Technique

Now that we have defined what we are looking for (a set of parameters θ^* which scores best using our $fit(.)$ function), we need to define how we are 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. How we explore the parameter space is defined by 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. 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 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 is reasonable to evaluate a candidate set of parameters which are somewhat similar, 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 explored regions of the parameter space may leave other, possibly higher-performing areas of the parameter space completely unexplored. It could be in one of these unexplored regions where the true, best-fitting parameters lie. 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 will 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

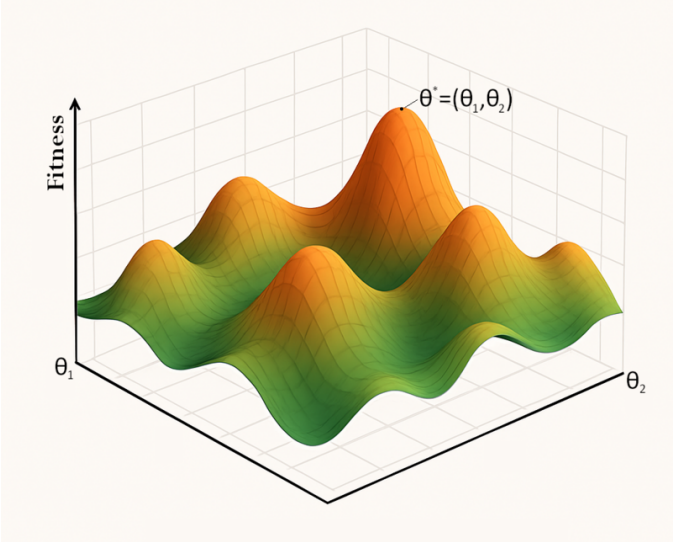
parameters (genes) from one parent and the rest from the other (emulating genetic crossover). Then their parameters have some chance of being randomly shocked (emulating genetic mutation). This new, resulting population is carried into the next round, and this process repeats until the search depth is 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 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 can be 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:

Figure 1: Example Fitness Landscape for a Two Parameter Optimization Problem



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. This would be computationally expensive at the very least, and if even one dimension of the parameter space is continuous, then it is simply impossible, as between any two points there are infinitely many more points to be evaluated. Therefore, features of the fitness landscape are often not obvious to the modeler a priori. This means 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 a much more relevant threat than if the landscape is single-peaked.

While there is no way to know what the most efficient way to search the space is for our problem a priori, we can investigate the level of performance a specification we have chosen is achieving once we have chosen one by running some tests 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 parameter estimates for 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. We will demonstrate how tests such as this can be run using AgentCarlo. Such tests are discussed in great detail in .

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:

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

where

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

Below, we detail how the estimation techniques described above can be easily applied using AgentCarlo.

6.5 Parameter Estimation in AgentCarlo

...

7 Bootstrapping Confidence Intervals

So far, we have 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 are sure that θ^* is the set of parameter estimates 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, our sample data, to learn about the population 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, what we want to establish is a kind of range of possible θ^*s that could result from repeating the same procedure on different samples.

A common approach to generate Confidence Intervals when using SMM requires iterated estimation of the Variance-Covariance matrix.[4] This method may not be ideal for many ABMs, however, as it relies primarily on the local slope in the fitness function (i.e. local approximate derivatives). This ignores the fact that there are often local optima in other regions of the fitness landscape which the search for best fitting parameters could easily return. This seems like a fairly big oversight in models where rugged fitness landscapes are possible. We argue for block-bootstrapping

[4] An accessible resource for this can be found at `https://opensourceecon.github.io/CompMethods/struct_est/SMM.html`

as one ideally suited alternative method which addresses this issue by directly re-estimating the parameters in slightly different scenarios to see how estimates vary when the sample data vary.

7.1 What is Bootstrapping?

Recall our aim is to understand how the sampling process affects our estimate θ^*. 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 the original sample data D with replacement. The argument goes that while we cannot generate new samples drawn from the population directly, our existing 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 5).

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, sample size, and type of data. Very narrow confidence intervals on a parameter θ_i^* can tell 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 Indicators 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. For example, a large confidence interval on a parameter could be signal that the model parameter is not *identified*, as your model may have multiple parameter specifications which achieve the same or very similar model output. This issue is discussed in detail in . 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 results in fairly different estimates. Once again, we discuss further the role noise from your model M_A and $search(.)$ function can play in producing estimate imprecision along with a novel diagnostic test to measure it in .

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 5. 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

$$\tilde{D_k} = \{b^1, ...b^G | b^m \overset{\text{iid}}{\sim} U(B)\} \tag{13}$$

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 will 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,

$$search_{\theta \in \Theta}(fit(\bar{Z}(Y_{M_A}, \theta, r), Z(Y_{\tilde{D_k}})), \delta) \rightarrow \theta_k^* \tag{14}$$

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 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,

$$\Delta^i = \{\varepsilon_1^i, ..., \varepsilon_K^i\} \tag{15}$$

where

$$\varepsilon_k^i = \theta^{i*} - \theta_k^{i*} \tag{16}$$

Importantly, we do not take absolute values of these differences, as we will 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 $\Delta_{Ordered}^i$ 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 $\Delta_{Ordered}^i$ and adding them to our best fitting parameter θ_i^*. Formally

$$C_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{17}$$

where

$$m = \lfloor K\frac{\alpha}{2} \rfloor + 1 \tag{18}$$

and

$$n = \lceil K(1 - \frac{\alpha}{2}) \rceil \tag{19}$$

$\lfloor . \rfloor$ 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 $\Delta_{Ordered}^i$. Further, the confidence interval for θ_i^* could be computed as $C_i = [\theta_i^* + \varepsilon_{6th}^i, \theta_i^* + \varepsilon_{195th}^i]$. Also note that while it may appear odd to add the error ε_{6th}^i for the lower bound of our confidence interval, ε_{6th}^i 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.

$$C_i = \theta_i^* + \varepsilon_{mth}^i \tag{20}$$

where

$$m = \lfloor K\alpha \rfloor + 1 \tag{21}$$

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. We describe how bootstrapping critical values for ABM parameters can be easily executed using AgentCarlo.

7.4 Bootstrapping Critical Values in AgentCarlo

...

7.5 Runtime Concerns

Runtime can be a serious concern in general when running or fitting ABMs, depending on their scale or the number of parameters needed to be estimated. Perhaps the greatest detriment of bootstrapping for critical values is that it requires the re-estimation of model parameters K additional times. Thankfully, there are a number of ways to address this. First, each of the K estimates (in Step 2) can be computed in parallel and then collected at the end to generate the confidence intervals. This can allow you to split total computational time across multiple machines/nodes. Second, if need be, a smaller K (k=50 for example) can also be chosen. Note however that this changes what the effective rejection rate is. Finally, we note that as we see computational power continue to grow (consistent with Moore's Law [Moore, 1965]), the computational burden imposed by this process will shrink.

8 Monte Carlo Simulations using AgentCarlo

In addition to the methods discussed above, AgentCarlo also provides a suite of tests that use Monte Carlo simulations to establish some properties about our ABM parameter estimates and the estimation process used to retrieve them. Such tests are important for establishing that the retrieved estimates are reasonably accurate, precise, and unbiased.

8.1 What is Monte Carlo Simulation

If we knew the true values of the DGP_{Data} used to generate our data, θ, (assuming the parameters of DGP_{Data} are analogous to the parameters of our ABM), then

establishing if our estimates are accurate, precise, and unbiased would be fairly trivial. We could simply compare our estimates $\hat{\theta}$ and our confidence intervals C to the true values θ. If we knew θ though, then we would have no use for our estimation process to being with.

What we can do, however, is to generate data using our ABM (a.k.a DGP_{Model}) by picking a set of parameter values θ to plug into our model and recording its inputs and outputs as data D. Then we can simply fit that 'simulated' data using our estimation process as usual to retrieve $\hat{\theta}$ and, if so desired, their corresponding confidence intervals C. Finally, we can compare our outputted estimates of the simulated data directly against the known values θ which were used to generate that simulated data.

Variations of this process can be used to explore a number of questions. We discuss this process in greater detail in . Below, we detail how AgentCarlo can be used to test for the degree of estimate imprecision and inaccuracy. This is particularly important for establishing that the parameters of the model are reasonably identifiable. We also include a test which can demonstrate the degree to which estimates of each parameter are biased. Finally, we incorporate a test first introduced in which can be useful for disentangling how much noise in model output and the estimation process contributes to estimate imprecision.

8.2 Monte Carlo Tests in AgentCarlo

...

9 Next Steps

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 have provided a practical guide for analysts to bring to a broad class of computational models to data. We are in the process of developing a fairly easy-to-use Python package for ABM estimation, AgentCarlo, which will provide a low-cost method for applying the methods described above and integrating it into this document.

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.

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

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

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

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

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.

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

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

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.