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

October 30, 2024

Abstract

While 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 computational models across many disciplines, the need for accessible and econometrically sound methods for estimating these models in one's toolkit has never been greater.

Taking the view that ABMs are in many ways analogous to structural equation models, we detail a fairly generalizable estimation framework 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 optimization), 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 (including simulations for decomposing sources of estimator imprecision).

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 as early as 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, no easy closed form characterization of the model's dynamics could be found, and so to characterize the model, computations over the board were performed (by hand), aggregated, and reported. 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 has been the subject of debate in economics.

While many utilizations of such 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. While some great work has been done on developing elements of 'Agent-Based Econometric Methods' (see), there remains a serious lack of established (and accessible) 'best practices.' What we need is a start-to-finish practitioner's guide which lays bare a methodology that is accessible and is grounded in existing econometric methods.

With this paper, we aim to do just this. We propose a fairly generalizable methodology for bringing ABMs to panel data with two goals in mind. First, this paper summarizes many fundamental concepts surrounding ABM estimation, allowing it to serve as a 'starter guide' for any interested analyst with an ABM. Second, we detail constructing critical value(s) via block-bootstrapping which, to the best of our knowledge, is fairly under-explored in the context of ABMs (Guilfoos and Pape [2016]). Further, we introduce a Monte-Carlo simulation which can clarify the magnitude to which different sources contribute to estimate imprecision.

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 provide 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 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 to data is to view ABMs through the lens of structural equation models. 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 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

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

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 of ABMs which serve often as strengths also present unique sets of challenges during estimation. First, they 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 in unique and possibly more realistic ways (e.g. rational decision making economic models). 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. 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 will be particularly useful for creating blocks of data we can use to bootstrap for critical values. Lastly, path dependence combined without the assumption that errors additively affect the system 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 we want to compare the model to data. 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. Importantly, we ought to make sense of what exactly we would learn by estimating such a model in the first place. 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 modify 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 which 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 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 will be compared to the output in data.
- A 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 your summarized model output and the 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 the M_A is consistent with the 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 with what precision we the parameters of the model are estimated and further, 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 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 the method we're recommending in later sections.

[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.

4 On Data

For the purposes of applying this method, we'll assume the modeller 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'll 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'll 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 for non-random reasons. This can introduce selection bias issues. 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're trying 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] and to name a few).

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 then, the term 'micro-foundations' is simply another name which economists have for 'micro-validity.'

While the validity concepts mentioned above may seem complete, we must also consider how the DGP corresponding to the data we have on hand DGP_{Data} corresponds to both the our model DGP_{Model} and the DGP which guides outcomes for the population of interest in the real world $DGP_{Reality}$. 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 DGP_{Model} and DGP_{Data} correspond, while external validity focuses more-so on the degree of 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 differs 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, while not mentioned explicitly, can be achieved through reasonable levels of both internal and external validity. If there is a correspondence between DGP_{Model} and DGP_{Data} and there's also a correspondence between DGP_{Data} and $DGP_{Reality}$, then there must be 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 (model-to-reality) validation is to lay bare and make accessible your model design and results. Taking this 'Hands-Above the Table' approach with both the models 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. Often times, ABMs are used 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 using a similar construction in the special case, the ABM in question is equally Micro and Macro valid 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 reality 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 prediction than to explanation 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 in hand, denoted M_A, 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. Inherently, this means we need:

- 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 summarized ABM output Y_{M_A} and what we observe regarding the relevant variable in data Y_D.

6.1 Defining a Summary Function

As noted above, a summary function is a wrapper around your model which makes your model output compatible to data. So why might it not be already?

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 highly depends on the model in question, but often it can take 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.

For example, 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. At the end of each run, the model results in some fixed configuration of agents positioned on a lattice (under non-trivial cases). To say something about macro-level segregation, each resulting configuration needs to be summarized in some way, i.e. a mapping from the configuration to 'macro-level segregation' needs to occur. In this example, for a configuration, macro-level segregation is summarized by the average portion of same-type neighbors each agent has. Easily we can imagine other such summary functions, the simplest modification of which could be to capture both the average and variance in same type neighbors had in the final configuration. Also, if we're interested in phenomenon over time, as we are when using panel data, then we should plot this same-type neighbor ratio over each step of the model.

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, you 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 simple summary function $S(.)$ which returns our expected output may take the form:

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

Note, however, that the summary function $S(.)$ can also return other information as well. 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 at the group level. In such cases, instead of returning a summary statistics for all agents in the model, summary statistics can be calculated individually for each group of agents.

6.3 Defining a Fitness Function

A fitness function $fit(.)$ defines a metric for evaluating how well your model is performing. 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 (commonly referred to as *fitness* if we are maximizing or *loss* if we are minimizing). On what basis, then, should we grade our models 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.

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 score. 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 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).

How is the space searched? 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.

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'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.

How do these methods work? For a taste of how these play out, we'll briefly discuss a few commonly used methods: *Grid-search*, *Hill-climbing*, *Simulated Annealing*, and the *Genetic Algorithm (GA).* Importantly, these methods have available libraries in commonly used programming languages (e.g. Python, C++) and most are natively supported in NetLogo as well.

A grid-search is a somewhat brute-force method each round of which has three steps. First, sample a 'grid' of equidistant points from the parameter space. Second, evaluate each of those points and find the one with the highest fitness. Third, shrink the search window of the parameter space to be around the best performer. This is repeated a number of times equal to the *search depth*, with the window of parameters considered shrinking after each depth. While this method casts a fairly

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

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, so using this method will result in the same estimates on the same data, which may prove desirable as we'll discuss later.

Hill-climbing is another method which starts in a particular place on the parameter space (randomly chosen). Each round a number of nearby points are chosen and evaluated along with the starting point. The best performer becomes the new starting point and the process repeats until the search depth is reached or some measure of convergence is met (e.g. x failures to move in a row). This method can be sensitive to the starting position chosen and can settle fairly easily on local optima since it has no way to go back down any hill it starts climbing. Some modifications include running it multiple times from different starting points and simulated annealing, which some probability to move to worse performing points (down-hill) with some probability that shrinks over time.

Finally, the genetic algorithm. This interesting method of optimization barrows from nature the ideas of natural selection and sexual reproduction within populations of fairly well performing candidate solutions (parameter sets). First, the 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 fairly robust and serves as a bread-and-butter optimization technique for all kinds of complex problems, though it can be computationally expensive for some problems.

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 the *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. Some examples of these can be found below:

[TODO: Add Finess Landscape Pictures Here]

It is rarely, if ever, feasible to view the fitness landscape for our problem of interest, as that would require exhaustive evaluate all possible combinations of our parameters. Therefore features of the fitness landscape are often not obvious to the modeller 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 accurate the search method is at returning estimating 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 the 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 the 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 $search(.)$ 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 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 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^{i*}_k, for each of the parameter sets in Z to compute errors ε^i_k. Formally,

$$\Delta^i = \{\varepsilon^i_1, ..., \varepsilon^i_K\} \tag{13}$$

where

$$\varepsilon^i_k = \theta^{i*} - \theta^{i*}_k \tag{14}$$

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, but also particularly so for many ABMs. Perhaps the greatest detriment to this method is that it relies on many reruns of the model. For example, above we mentioned that to generate 95% confidence intervals, one should fit their model 200 additional times on resampled datasets. While this may not be feasible for all models, there are a number of things that can be done to improve the feasibility. First, you can always reduce K to K=50 for example, though this will shrink the true rejection rate as the confidence intervals will be a bit larger. Second, each of your K fits on resampled data can be run in parallel, as they are completely independent during the fitting process (Step 2 above). Thus, K=50 could take the form of 5 separate submissions to a computing cluster to execute 10 estimates each. Lastly, we hope that, consistent with Moore's Law [Moore, 1965], as we see computational power continue to grow, the computational time required to perform this process will shrink.

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, and for which noise likely enters the model non-trivially, and similarly an estimation technique $search(\delta, .)$ that itself is stochastic and does not guarantee optimal solutions, one should be cautious in assuming that θ^* and C are sensible. In particular, we want to know if we can estimate reasonably accurate, fairly unbiased estimates of θ^*. We'd also like to know the degree to which imprecision of our estimate θ^* (which is demonstrated by our estimated confidence intervals C) reflects the actual sensitivity of our estimated parameters to sampling variation, and if we can improve on that imprecision. Monte-Carlo simulations can serve an essential role in establishing such properties. For the remainder of this section, we discuss Monte-Carlo simulation in general and then propose candidate simulations which can be run to address each of the questions introduced above.

8.1 Monte-Carlo Simulation - What and How

Monte-Carlo Simulations are, broadly defined, simple computational models which are used to establish properties about some distributions (or optimization technique) 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 about $search(.)$ as returning one such θ^* to us from some unknown, underlying distribution over all possible θ^*s. Our aim in this section is to learn about this distribution. So how can we do this?

At the core of these proposed simulations is the supposition that there is some true, underlying data-generating process in the real world which our data came from, DGP_{Data}. When we bring our ABM to fit this data, we are taking as given that our model M_A is a reasonable approximation of this DGP and aim to estimate its parameters.

Recall that our aim is to establish that our estimated parameters θ^* are reasonable. Now if we knew the true parameters θ of the real DGP, 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. While we obviously don't know θ or the functional form of DGP_{Data}, we can do something quite similar.

Let's suppose for a moment that our model M_A is precisely the true DGP, DGP_{Data}. Next, let's choose some parameters θ randomly 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 simply feeding in values for X (which can be chosen randomly if need be) and recording model output Y. Importantly, this data was generated using parameter values θ which are known, because we chose them. Formally,

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

where values in X which are given exogenously (i.e. are not determined within the model) are randomly drawn from some distribution.

Now we can estimate the parameters θ^* on that same model M_A using $search(.)$, treating this simulated data $\hat{D}$ as real data. Formally, paralleling what we did in equation 3,

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

Finally, knowing the true, chosen values of θ and now having estimated parameters θ^*, we can simply compare the two.

Much of the remainder of this section delves into different such comparisons we can do and how they may prove useful. But first, how do we interpret the results of this process given we're treating our model M_A (a.k.a DGP_{Model}) as DGP_{Data}? Can they tell us anything about how well we can estimate DGP_{Data} when we're not even using the DGP_{Data} in the simulation? One position an analyst can take is to make the case that the model M_A is in fact a reasonable substitute. In reality, this is the position we take when we aim to estimate a structural model in the first place. This argument rest strongly on how solid of a case has been made for model validity. Another more conservative way to view this exercise is that it is measuring estimator performance under the 'best case scenario'. If the model cannot perform reasonably well when fitting the exact same, known DGP as our model DGP_{Model}, then what hope does it have at recovering the true parameters of a likely somewhat different, unknown DGP, DGP_{Data}? From this view, this is a necessary test to establish reasonable performance, but is not sufficient.

8.2 About Estimate Accuracy and Bias

Recall our model M_A is designed to take some input (potentially over time) X and some parameters θ, and from that, returns some output Y over time. When we try to estimate the parameters of our model, though, we are treating our model as a good approximation of the true DGP and aim to use X and Y to back out what the underlying parameters θ are. Aiming to solve this *Inverse Problem*, we introduced $S(.)$, $Agg(.)$, $fit(.)$ and $search(.)$ functions above in section 6 which are used together to retrieve our estimate θ^* of the true parameters θ. It is not clear, however, that the transformation of inputs X into outputs Y should always correspond to a unique set of parameters θ. If multiple, fairly different parameter sets are all similarly likely to produce the same output Y given the same inputs X, then observing X and Y alone will not be sufficient to back out what the underlying parameters θ are. If for all possible output Y from our model M_A given inputs X there is a unique parameter set θ which is most likely to produce said output given X, then we would say the model is *Identified*. This is one of a number of possible reasons why estimates θ^* could be inaccurate which is not unlikely in the context of ABM estimation.

In some cases, failure to meet this requirement may be fine if the multiple parameter sets which can produce the same outputs are reasonably close. In the Schelling Model for example, an intolerance level of 26% and 27% should produce the same distribution of output if performed on a lattice, since agents can only have at most 4 neighbors. The distinction between both can be seen as inconsequential though, since they live close enough in the parameter space that they share both a fairly similar interpretation and would have similar potential policy implications. Hence, we use the phrase *Reasonably Identified* to relax Identification to allow for multiple parameter sets to correspond to an output Y given inputs X so long as the parameter sets are reasonably close to one another and similar in interpretation. By establishing a reasonable degree of estimation accuracy, we demonstrate that our model and estimation technique together are capable of returning estimates θ^* which are close to and consistent with θ, sidestepping model identification issues and provides some basic level of assurance of performance.

A number of accuracy measures can be considered. For the sake of this paper, we entertain a measure meant to capure the 'average distance' between estimated parameters θ^{i*} and the underlying θ^i. Formally,

$$AvgDist^i = E(|\theta^{i*} - \theta^i|) \tag{22}$$

where —.— is the absolute value.

Note that this is calculated separately for each parameter, as we want to see how accurate our estimate for each parameter is in turn. If this measure is close to zero for a parameter, than it indicates that we can perfectly recover the initial parameter θ^i consistently. This will likely never be the case, but we're hoping that on average, we're not too far off.

While the estimate accuracy measure considered above captures how far we are from the true parameter value on average, *Bias* aims to uncover if that distance is on average more likely to be above or below θ_i. Formally, this is given as

$$Bias^i = E(\theta^{i*}) - \theta^i \tag{23}$$

We say an estimator θ^{i*} is said to be *Unbiased* when $Bias^i = 0$. Establishing that the estimator is unbiased, or at least that bias is small for each parameter in θ^8 is important, as it indicates that your estimate tends to be correct on average.

8.3 A Test for Model Accuracy and Bias

8.3.1 1. Randomly Draw θ_ms

For this first Monte-Carlo simulation, we'll randomly sample M sets of parameters $T = [\theta_1, \ ..., \ \theta_M]$, each of which are drawn randomly from the parameter space. Formally,

$$T = \{\theta_1, ...\theta_M | \theta_m \overset{\text{iid}}{\sim} F(\Theta)\} \tag{24}$$

where $F(\Theta)$ is some probability distribution over the parameter space Θ. One very reasonable choice for F might be to draw each parameter in θ_m uniformly from the parameter space.

8.3.2 2. Generate Simulated Datasets for each θ_m

For each parameter set θ_m in T, we're generate J corresponding simulated datasets $\{\hat{D_{m,1}}, ..., \hat{D_{m,J}}\}$. Each of these simulated datasets $\hat{D_{m,j}}$ are sets of possible outcomes we could observe if M_A was the true DGP and θ_m was the corresponding true, underlying parameter values. Simulated data can be created using equation 20 from above. We rewrite this with slightly new notation for our context below as:

$$M_A(\theta_m, X) \rightarrow \hat{D_{m,j}} \tag{25}$$

8.3.3 3. Estimate θ^*_m for each Simulated Dataset

Next, for each simulated dataset $\hat{D_{m,j}}$, we estimate θ^*_m to see if how well we can back out θ_m. θ^*_m is estimated by simply applying our *search*(.) method on the corresponding simulated data $\hat{D_{m,j}}$ as is shown in equation 21. From this process, we should have J estimates $\{\theta^*_{m,1}, ...\theta^*_{m,J}\}$ corresponding to each parameter set θ_m.

8.3.4 4. Evaluate Estimates and Calculate Biases

Finally, for each parameter i in each parameter set θ_m in T, we calculate how off the model is on the average distance and bias for each parameter i using the corresponding set of estimates $\{\theta^*_{m,1}, ...\theta^*_{m,J}\}$. Formally,

$$AvgDist_m^i = \frac{1}{J}\sum_{j=1}^{J} |\theta_{m,j}^{i*} - \theta_m^i| \tag{26}$$

$$Bias_m^i = \frac{1}{J}\sum_{j=1}^{J} \theta_{m,j}^{i*} - \theta_m^i \tag{27}$$

By computing bias and average distance for M different regions in the parameter space (i.e. for different parameters θ_m), we can see how consistent these measures are across that space.

8.4 Addressing Simulation Results

To establish reasonable estimate accuracy, we'd need to observe $AvgDist_m^i$ as fairly small across units of i and m. If this is not the case, then it is possible that your model is not reasonably identified. Another cause, however, could be that your $search(.)$ method is fairly noisy. If this is the case, then the following Monte-Carlo simulation is further motivated to investigate the size of such noise and, importantly, if changes to $search(\delta, .)$ can reduce the noise component.

Similarly, we'd like to see estimation bias as fairly small, otherwise, on average, we will get estimates which are systematically higher (or lower) than they should be. In such cases, adjusting the fitness function $fit(.)$ to further prioritize first order fit may help reduce bias. It may also be the case that, even with a sizable bias, the parameter estimates retain some interpretability as a bound on the effect size. For example, if the case being made is on the size of some effect of some parameter Z on Y, and θ_Z^* is negatively biased, then we can say we expect Z has at least a θ_Z^* effect on Y.

8.5 About Sources of Estimate Imprecision

While the confidence intervals C we generate using bootstrapping can be used for hypothesis testing in the same way one normally would in other contexts, our outputted Cs are slightly different in interpretation. In particular, they capture some additional sources of variation beyond sampling variation. The first source of noise comes from the stochastic nature of our $search(.)$ operation which, even if it is given the same problem, may converge on a different solution by virtue of being a partially random process. Our second source of noise comes from our model M_A itself which likely has some stochastic component. Even though we aggregate the summarized model output across a number of runs, this aggregated model output may still be fairly noisy. The noise from these two components can cause our $search(.)$ method to face fairly different summarized output, even when evaluating the same candidate set of parameters, yielding potentially fairly different fitness evaluations.

This problem can be addressed two-fold however. First, this additional variation means our confidence intervals C 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 model M_A and the search process $search(.)$ play a role in estimate imprecision, and the degree to which this imprecision can be reduced via changes in $search(.)$ or $Agg(.)$.

Why does this matter? Recall above in section 7.2 we established that confidence intervals can signal how precise our parameters are estimated (i.e. how robust our estimates are to sampling variation) and can be used for hypothesis testing to establish if our estimate is significant (i.e. we're fairly confident there's a non-zero effect of this parameter). It is possible that $search(\delta, .)$ and M_A contribute substantially enough that our confidence intervals C are completely uninformative (as they span nearly the entirety of each dimension of the parameter space) and our effective rejection rate of our null hypothesis become nearly 0. It's much better to just measure the size of this noise in isolation and then explore how adjustments to $search(.)$, our search parameters δ, and our model runs for aggregation r can reduce this noise if at all.

8.6 Investigating Sources of 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 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 there is no variation in the sample data we've provided.

8.6.1 1. Find K Best Fits θ_k^* on Same Data

Taking the actual data D, conduct K searches for best fitting parameters θ_k^*. This is done in the same way as in equation 3. Using slightly different notation,

$$search_{\theta \in \Theta}(fit(Y_{M_A}, Y_D), \delta) \rightarrow \theta_k^* \tag{28}$$

8.6.2 2. Construct the Critical Value(s)

This is done precisely the same way as in section 7.3.3. Recall,

$$C_i = [\theta_i^* + \varepsilon_{mth}^i, \theta_i^* + \varepsilon_{nth}^i] \tag{15}$$

Each estimated confidence interval C_i demonstrates the size to which the parameter estimates vary given no such changes in sample data. For these confidence intervals to retain a similar meaning to their roots in other contexts with deterministic optimization techniques, then the range which these confidence intervals span should be fairly small.

8.7 Addressing Simulation Results

As mentioned above, if $search(\delta, .)$ or M_A are having a fairly large effect on your confidence interval estimates C, then some natural adjustments come to mind which may reduce this noise. For one, if the summarized model output $S(M_A)$ is fairly noisy, you can consider increasing the number of runs r you do and aggregate across. Similarly, if your search method $search(\delta, .)$ is similarly noisy, then some changes to the search parameters δ might be appropriate. Specifically, it could be that the method does not search long enough to converge on the result (in which case turning up the number of search iterations may help). It may also be the case that the search method does converge, but is susceptible to converging on local optima (in which case turning up parameters which control the degree of exploration may help). Making these adjustments and then re-running the test again can demonstrate clearly the degree to which these changes helped reduce the variation these components contribute.

9 An Application Example

10 Outlook and Conclusion

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.

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.