INDIVIDUAL AND SOCIAL LEARNING:
AN IMPLEMENTATION OF BOUNDED RATIONALITY FROM FIRST PRINCIPLES

by

Nathan M. Palmer
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computational Social Science

Committee:

_____	Dr. Robert Axtell, Dissertation Director
_____	Dr. Christopher Carroll, Committee Member
_____	Dr. Andrew Crooks, Committee Member
_____	Dr. Daniel Houser, Committee Member
_____	Dr. Kevin Curtin, Department Chairperson
_____	Dr. Donna M. Fox, Associate Dean, Office of Student Affairs & Special Programs, College of Science
_____	Dr. Peggy Agouris, Dean, College of Science
Date: _____	Fall Semester 2015 George Mason University Fairfax, VA

Individual and Social Learning: An Implementation of Bounded Rationality from First Principles

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Nathan M. Palmer
Master of Arts
Boston University, 2011
Bachelor of Science
Trinity University, 2005

Director: Dr. Robert Axtell, Professor
Department of Computational and Data Sciences

Fall Semester 2015
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my wife, who is my best friend and biggest supporter.

Acknowledgments

I would like to thank my committee – Rob Axtell, Chris Carroll, Dan Houser, and Andrew Crooks, for their exceptional insight and guidance through the process. I would not be the researcher I am now without them or my colleagues and classmates at George Mason and other research organizations. I have too many to thank, and hesitate to list all here as I will certainly leave some off by pure oversight.

Table of Contents

List of Tables

List of Figures

Abstract

INDIVIDUAL AND SOCIAL LEARNING: AN IMPLEMENTATION OF BOUNDED RATIONALITY FROM FIRST PRINCIPLES

Nathan M. Palmer, PhD

George Mason University, 2015

Dissertation Director: Dr. Robert Axtell

This dissertation expands upon a growing economic literature that uses tools from reinforcement learning and approximate dynamic programming to impose bounded rationality in intertemporal choice problems. My dissertation contributes to the literature by applying these tools to the canonical household consumption under uncertainty problem. The three essays explore individual and social approaches to learning-to-optimize and how these may be brought to data.

The first chapter unveils a novel learning-to-optimize algorithm which functions at the individual level: agents learn a near-optimal consumption rule from their own experience by using regret. This approach is new in the literature; it is closest to Q-learning but avoids the difficulties associated with applying Q-learning to economic problems. Numerical results demonstrate convergence to a stable neighborhood around the optimal policy. The second chapter extends the Allen and Carroll (2001) model in an alternative direction by incorporating social learning and introducing an new intuitively motivated estimator of the value function. Each extension individually reduces the time required for agents to learn near-optimal rules by an orders of magnitude while remaining simple to implement.

My final chapter outlines a general-purpose modeling toolkit: the Heterogeneous-Agent Computational toolKit (HACK). This chapter highlights the similarities between methodologies in agent-based modeling and more traditional heterogeneous-agent modeling and brings the dissertation full circle: the empirical methods employed in HACK will be used in future work to estimate agent-based models employing regret learning from the earlier chapter.

Chapter 1: Introduction

1.1 Motivation

Sims (1980) famously stated that non-rational behavior is a "wilderness:" there is only one way for economic agents to be "rational" but infinitely many ways to be non-rational. While this may be strictly true, some paths through the wilderness are more familiar and well-trod than others. Specifically, the economics literature has long employed the tools of optimal control, and particularly dynamic programming, to model agents as fully optimizing beings in dynamic and uncertain environments. However this optimizing behavior, particularly combined with "rational expectations," imposes extensive limits on the types of models which may be built to explain economic phenomena. This creates a tension: if structurally complex models are important for describing complicated economic behavior, agents cannot be given rational expectations with traditional tools. They may be given various ad-hoc behavior, but as Sims (1980) notes, the range of possible ad-hoc decision rules for dynamic behavior is infinitely wide.

1.2 My Contribution

This dissertation introduces a middle ground between optimization and boundedly rational behavior. Agents use a non-optimal rules to make decisions in a dynamic and uncertain environment, and after gaining enough experience they learn new rules. Learning is structured such that under *enough* experience, agents will learn rules which are in a neighborhood around the true optimal rule for the problem they are trying to solve. Even when their experience is short, however, they are always moving in a direction that appears best to them conditional on their experience.

To make this effort concrete, this dissertation focuses on the basic problem at the core of modern macroeconomics and finance: the household consumption-savings problem under uncertainty. Households have an uncertain income and can either save money at a risk-free return rate or spend it immediately on consumption. Consumption produces rewards for the household via a utility function; this utility function is used to create a lifetime objective function. Households are modeled by agents who seek to maximize the traditional objective function for this problem, but they are assumed to be incapable of finding the optimal solution instantaneously every period. Instead the agents use simple, non-optimal consumption rules to make decisions period-by-period. At particular intervals the households re-evaluate their rules: they reflect on their experiences thus far and then learn a better rules from experience. This can includes their own experience (Chapter 2) and the experiences of others.

On a technical level this learning is modeled after the most recent developments from the mathematical dynamic programming literature. These new methods are referred to as "approximate dynamic programming" in applied mathematics, and "reinforcement learning" in the computer science literatures (to be discussed further below). These tools have made slow headway in the economics literature in part because they have been formulated for problems with a structure different than those modeled by economists. Specifically they are often employed for dynamic problems with discrete state and action spaces, while many foundational problems in macroeconomics and finance require continuous state and action spaces. This dissertation addresses this concern by introducing a version of learning-to-optimize explicitly created for continuous state and action spaces and exploring how similar methods may be used in a social learning context.

This work is explicitly intended to provide a theoretical foundation for non-optimal agent behavior in both agent-based models and in more traditional macroeconomic models. I particularly want to capture the idea that economic agents would *like* to behave optimally, but they cannot for any number of reasons – perhaps they simply do not know how, or they do not have enough information, or the optimal solution is simply intractable because

the problem is too complex. I provide a rigorous description of agent behavior which is "stumbling towards the optimum;" with enough experience and time they can obtain near-optimal rules, but even without extensive time they settle into a stable distribution of rules near the optimal behavior. By using the same optimization framework as traditional models, I can directly calculate the welfare costs of this non-optimal behavior with respect to the optimal behavior.

1.3 Background

An agent displays rational expectations if they both optimize a well-stated objective functions derived from assumptions about their preferences and the expectations components of their behavior is in fact the true mathematical expectations over all states of their world. Alternatively, the agents' forecasting errors are white noise – their forecasts are never systematically wrong. The assumptions associated with rational expectations may seem excessive, but there are a number of good reasons they achieved widespread usage; Woodford (1999) is an excellent overview of the development of rational expectations and the historical context which gave rise to the idea. Very briefly, a "crisis of methodologies" occurred in the 1960s and 1970s during the period of the "great inflation" (Woodford, 1999). Among other things, macroeconomic models of the time modeled people as following very simple "rules of thumb" for behavior, which did not react sufficiently to changes in the structure of the macroeconomy. Problems inherent with these assumptions were revealed dramatically during the "great inflation" period, including deep technical problems which these assumptions implied for how macroeconomic models were estimated at the time. The interview Evans and Honkapohja (2005) in particular discusses a number of these technical issues and provides insight into the historical context behind the birth of rational expectations. Rational expectations arose as a promising solution to multiple problems at once, both theoretical and econometric and slowly dominated much of the literature; see Sheffrin (1996) and Mankiw (2006) for additional perspectives on this history. Sheffrin (1996) provides a

longer exploration of the topic including examples of the problems that gave rise to rational expectations (and as a bonus, Sheffrin (1996) provides historical details about the individuals behind the rational expectations revolution which is quite interesting in its own right.)

There are at least two difficulties encountered when using rational expectations to model agent behavior in a macroeconomy. The first is that human behavior may be captured quite poorly by an optimizing agent with correctly formed expectations. I will not focus on this first difficulty because many have discussed it at length; see for example Colander et al. (2008) and references therein. Instead I focus on the second difficulty: imposing rational expectations severely constrains size and complexity of general-equilibrium models, where general-equilibrium models are those in which agent behaviors affect the state of the model itself. Under various sets of stringent assumptions, so-called representative-agent models can be solved easily under rational expectations, particularly with the use of pre-built toolkits such as Dynare (Adjemian et al., 2011). Rational expectations heterogeneous-agent models, however, are very difficult to solve: extensive human capital accumulation is required on the part of the researcher and even then complete rationality in a complicated model may still be intractable (Allen and Carroll, 2001).

The solution proposed by agent-based modeling is to set up detailed models of markets and economies and give agents behavioral rules of thumb; see for example Colander et al. (2008) and Geanakoplos et al. (2012). Simon (1996) discusses at length how a complicated, difficult problem may be broken up into smaller, simpler problems by the environment in which the problem is solved – thus, the details and structure of a model may be such that agents following simple behavior easily find near-optimal or "satisficing" solutions which are "good enough."

The specter of the Lucas Critique, however, haunts modern research. The famous quote from Sims (1980) regarding the wilderness of bounded rationality is cited almost immediately by any macroeconomist who encounters non-rational agent behavior. The key question is always, "but where does that behavior come from? What objective does the

agent want to maximize?" A close second question is, "how do you bring these models to the data?" This dissertation aims to answer these questions in three essays, in ways which satisfy both the economist and the agent-based modeler.

1.4 What Has Been Accomplished

The dissertation contributes to the current literature by providing a rigorous description of plausible mechanisms by which an agent can learn the solution to a dynamic stochastic optimization problem from experience. This dissertation offers three analogies regarding how agents might learn from their own experience and the experiences of others. These analogies are then made rigorous by casting them in the framework of approximate dynamic programming. Numerical simulation results then show that these mechanisms converge towards the optimal solution in distribution as experience is taken to the infinite limit.

The first two essays expand upon Allen and Carroll (2001), an early model from the economic literature on learning to optimize in dynamic optimization problems. In particular agents learn by using a fixed, non-optimal consumption rule for a number of periods (denoted "D") before stopping to "take stock" of their experience and update their consumption rule based on their own experience or the experiences of others. I do not take a stance on why agents stop and re-evaluate their consumption function after D periods. Instead in this preliminary examination I simply analyze what effect different lengths of D will have on their ability to learn a new rule, and leave the choice of D exogenous for now. In future work I intend to expand the algorithm to make D endogenous.

Chapter 2 introduces a novel learning-to-optimize algorithm which functions at the individual level: agents learn a near-optimal consumption rule from their own experience by using "regret." This approach is new in the literature; it is closest to Q-learning but avoids the difficulties associated with applying Q-learning to economic problems (See Section (2.1) for a discussion of differences between Q-learning and Regret Learning). Numerical results demonstrate convergence to a stable neighborhood around the optimal policy. Chapter 3 extends the Allen and Carroll (2001) model in an alternative direction by incorporating

social learning and introducing an new intuitively motived estimator of the value function. Each extension individually reduces the time required for agents to learn near-optimal rules by an orders of magnitude while remaining simple to implement.

The surprising results of these two chapters are that learning a near-optimal rule can be accomplished with explicitly biased estimates derived from a single agent's experience. As described in greater detail in Section (2.4.1) below, it is straightforward to formulate a simple Monte Carlo estimator for the optimization problem faced by an agent. However as discovered by Allen and Carroll (2001), being able to form this estimator is not enough – without a highly efficient search or learning method, it can take millions of periods to approach a near-optimal rule. Regret learning demonstrates that a highly efficient search method does in fact emerge from a single agent's experience that is highly biased (as a single time-series draw from the data generating process). This is discussed further in Section (2.1.2) and Chapter (2) as a whole.

Finally, the Heterogeneous-Agent Computational toolKit (HACK) is joint work developed with Chris Carroll and is discussed in Chapter 4. It aims to bring structured code to the world of heterogeneous-agent economic computation. Heterogeneous-agent economic models, in particular as discussed in Chapter 3, are one of the classes of models in traditional economics which are close to agent-based models. It is the hope of this author that this toolkit can act as a bridge between the disciplines.

The contribution of the learning-to-optimize literature in general and this dissertation in particular is the note that agents can obtain optimal solutions from learning in real time. This is not a result which is clear from observing the first principles of dynamic optimization theory. Even when an agent only has access to a single stream of experience, as in Chapter 2 in this dissertation, and thus their own learning is highly correlated, they can nonetheless find an near-optimal policy function. To say it another way, estimating a value function from very little experience produces a highly biased value function – and yet learning a highly biased value function can still lead to learning a nearly correct consumption function.

Finally, the solution method presented in this dissertation is explicitly intended to be an

engine to drive development of both agent-based models such as Geanakoplos et al. (2012) and large-scale heterogeneous-agent macroeconomic models such as Peterman et al. (2015).

1.4.1 Welfare Costs of Non-Optimal Behavior

As soon as an economic agent follows non-optimal behavior, the immediate question arises: what are the welfare costs of this behavior with respect to following the optimal solution? In this way, optimization can act as a benchmark or a metric for measuring "distances" between different sets of non-optimal behavior. If a non-optimizing behavior can be formulated such that it produces a value function, the welfare costs of this non-optimal behavior can be measured. This is one additional reason the learning presented here formulated as approximations in the dynamic programming framework – it provides an immediate way to create measures of welfare costs related to this behavior.

I use the welfare costs described by Allen and Carroll (2001): the "sacrifice value" of following any non-optimal rule for the rest of one's life. This is described in greater detail in the Chapters 2 and 4, specifically Section (2.2.3) as well as in Allen and Carroll (2001) but I will briefly describe it here as well. Assume an agent is perfectly rational and using the optimal rule. The sacrifice value for a particular non-optimal rule is the amount of money the rational agent would be willing to pay to *not* switch from the true optimal rule to that particular non-optimal rule. This is expressed in fractions of expected annual income and acts as a metric to measure distances from any non-optimal rule the agent learns to the optimal rule.

There is actually a deeper question at hand when one asks the welfare cost of a learning algorithm. The sacrifice value noted above is a value that assumes an agent follows the non-optimal rule for the rest of their lives – this is what the value function represents, and the value functions are used to calculate the sacrifice value. This is an important and reasonable first pass. However in learning agents are likely not using the same rule for the rest of their lives. The value function associated with the *entire learning process* is almost certainly different form that of following a specific rule. This "meta" value function of

learning may not even be monotone, which is a requirement for measuring unique sacrifice values. At the intuitive level, if it takes a non-trivial set of periods in a finite life to try another policy, the opportunity cost of agent time is the best policy the agent has seen so far – presumably the current one. How does the agent decide when experimenting with a new policy is not worth it? There's a risk aversion decision buried in this decision which has not been explored extensively in the economic literature, in part because it requires agents to be using non-optimal rules to guide their behavior.

This is a question has been understood for a long time in the reinforcement learning literature, to be discussed below. In that literature this question is known as the "exploitation/exploration trade-off," and is most cleanly illustrated by a set of simple problems called "bandit" problems. See Sutton and Barto (1998) and Auer et al. (2002) for an excellent description of both the exploration/exploitation trade-off and its illustration in bandit problems. I do not deal with this deeper question of the welfare costs of learning here, instead leaving this to future work. I will simply note that the exploration / exploitation trade-off is a key fundamental question as soon as one moves away from instantaneous optimization. I strongly suspect that many behaviors, such as satisficing, have a basis in this trade-off.

1.5 Related Literature and Alternative Approaches

There is a tremendous literature on "learning" in economics, much too broad to be reviewed here. The specific type of learning which this dissertation addresses is learning the optimal solution to a dynamic optimization problem. This learning can take at least two forms: statistical learning about distributions followed by optimization conditional on beliefs from learning, such as in Chamley (2004), or at the macroeconomic scale as in ordinary least squares (OLS) learning in Evans and Honkapohja (2001). An alternative is learning to optimize directly by searching a space of non-optimal policy function for something near the optimal solution. Three of the best examples of this latter type of learning can be found in Allen and Carroll (2001), Howitt and Özak (2014), and Özak (2014). Allen and

Carroll (2001) lays out the basic consumption-savings problem under uncertainty, which is at the core of all modern macroeconomic and finance models. They observe that a first-order linear approximation to the *form* of the true optimal rule produces a simple and intuitive consumption function, which agents can learn about from experience. Their work produces a positive and negative result: agents can learn a near-optimal consumption function from averaging over enough experience, but enough experience must be in the hundreds of thousands or millions of periods. Since their model is parameterized such that a period is a year, this is very far from appearing practical for use in any model, either traditional or agent-based. Howitt and Özak (2014) and Özak (2014) address the same problem introduced by Allen and Carroll (2001), but employ learning based on the Euler equation. Their agents must be somewhat more sophisticated than those of Allen and Carroll (2001), specifically in the sense of being able to solve envelope conditions of their optimization problem, but the payoff is much faster learning, closer to 50-100 periods.

1.5.1 Literature in Computer Science and Applied Math

The algorithms used by both Allen and Carroll (2001) and Howitt and Özak (2014) are examples of what is known as "reinforcement learning" in the computer science literature, or "approximate dynamic programming" in the operations research and applied mathematics literature. Both names communicate important aspects of this methodology: "approximate dynamic programming" emphasizes that this is a generalization of dynamic programming, employing much of the same theoretical framework to find optimal policy and value functions associated with a given Markov Decision Process (MDP). Bertsekas (2012) and Powell (2007) explicitly outline the relationship between reinforcement learning and traditional dynamic programming. More recently, Bertsekas (2013) casts traditional and approximate dynamic programming under a single unifying analytical framework. Bertsekas and Tsitsiklis (1996) outline much of the foundational theory which ties online optimization methods to dynamic programming. Approximate dynamic programming, however, does not require

information about the transition probabilities of the MDP to achieve the same results. Initial arbitrary policy and value functions are incrementally updated by a decision-maker based on experience; in a stationary environment, these functions converge to the optimal policy and value for the given MDP. The key insight is that (a) optimization programs can be solved asynchronously, that is, with an appropriately random order of value and policy function updates during the dynamic program iterations itself (that is, the fixed-point calculation steps, discussed in greater detail below), and (b) "online" experience can act as a type of Monte Carlo implementation of the asynchronous dynamic programming optimization. Thus agents can, quite literally, learn to optimize from experience alone, in a well-understood analytical sense. This broad description also underlines the reasoning behind the term reinforcement learning: the optimal policy and value function are learned, incrementally, from experiences which "reinforce" the values of the best actions. Finally, Sutton and Barto (1998) outlines much of the foundational results and literature from the computer science perspective.

The differences between the approximate dynamic programming literatures and the reinforcement learning literatures extends further than the names alone. They use different proof methods as well. The approximate dynamic programming framework as described by Bertsekas (2012) and Powell (2007) use contraction mapping as main proof method, similar to the traditional dynamic programming literature. This yields useful analytical results for some of the basic methods, such as bounds on error and convergence times. This analytical framework is Lucas Jr and Stokey (1989). The reinforcement learning framework, as described in Sutton and Barto (1998), uses the stochastic approximation methods originating from Robbins and Monro (1951) as the main analytical tool, which can be difficult to interpret from a traditional economic perspective.

Finally, in the computer science literature, different formulations of "regret" are used to judge how well an online reinforcement learning algorithm accomplishes a task of learning-to-optimize. Some definitions and theoretical results related to "regret" and "regret bounds" can be found in a number of papers in this literature, including Jaksch et al. (2010), Auer

et al. (2002), and Mannor and Tsitsiklis (2004). I use the term "regret" to describe a different process than the one used in these literatures. Namely, I use it to describe the way in which agents in my model reflect on their own past experience and make better choices (in a utility-maximizing sense) once they have learned the value of their previous choices.

1.5.2 Literature and Alternative Approaches in Economics

There is a rich history behind the development of reinforcement learning and approximate dynamic programming. Sutton and Barto (1998) trace out a number of parallel threads of this history, running through psychology, biology, computer science, operations research, and even neuroscience. Tesfatsion and Judd (2006) provides an excellent outline of early uses of reinforcement learning in economics. More recently, a handful of authors have explored reinforcement learning applied to intertemporal consumption-savings problems. Learning about intertemporal choice poses a particular challenge: obtaining enough information to estimate value and policy functions accurately requires many draws of time series. Since the time series are generated by agent experience, it can be quite difficult to obtain the number of draws needed. Lettau and Uhlig (1999) is one of the earliest such efforts; the authors use a classifier system (an early form of reinforcement learning) which chooses between an optimal rule and a version of the spendthrift ("consume everything") rule. Their model learns the non-optimal rule under a wide range of conditions, and the authors caution readers that households in practice may not learn optimal rules for similar reasons. Interestingly, Başçı and Orhan (2000) revisit Lettau and Uhlig (1999) results and find that agents are once again able to distinguish the optimal rule from non-optimal rules when "trembling hand" experimentation is allowed.

As noted above, Allen and Carroll (2001) use a clever parameterization of the consumption function for a simple consumption-savings problem and employ a Monte Carlo learning estimator to chose the best policy on a fixed grid of policies. They find that (a) agents can find a near-optimal rule given enough time, but (b) enough time can be millions of periods, rendering the learning not useful for practical time parameterizations. They suggest that

one solution to this may be social learning, which is explored further below. Howitt and Özak (2014) and Özak (2014) address the same consumption problem as Allen and Carroll (2001), but using a clever form of policy-gradient learning which utilizes marginal utilities to update a consumption function. Policy-gradient learning is a form of reinforcement learning which attempts hill-climbing on an appropriately specified value function surface; see Sutton and Barto (1998) for an intuitive overview. As noted above, Howitt and Özak (2014) and Özak (2014) obtain agents who can find a near-optimal rule quickly but must know more about the structure of the problem than agents of Allen and Carroll (2001).

The above models are largely cast in a partial-equilibrium framework, in which agents actions do not affect the aggregate states of their models. One of the main reasons that learning-to-optimize is interesting is that it may influence aggregate dynamics. One of the original papers related to this is Krusell and Smith (1996), which predates their famous Krusell and Smith Jr (1998) paper. In Krusell and Smith (1996), they examine the general equilibrium effects of agents choosing between using the true optimal solution and various rules of thumb, when the true optimal solution has some small cost. They find that very small costs to optimization, less than a tenth of a percent of per-period consumption, can cause agents to choose to use rules of thumb, which in turn change the statistical dynamics of the aggregate system in non-trivial ways. Evans and McGough (2014) address a similar problem as that of Allen and Carroll (2001), but in a general equilibrium context. Like Howitt and Özak (2014), their agents must know something about the first-order conditions of their optimization problem, and use these to learn about the shadow process of their choices. As with Krusell and Smith (1996), they find that the aggregate dynamics, particularly the transition dynamics, are greatly affected by learning.

An alternative application of learning-to-optimize is finding optimal solutions when traditional techniques are intractable. Examples of this include Hull (2012) and Jirnyi and Lepetyuk (2011). Hull (2012) constructs an overlapping generation model with 60 rolling cohorts of agents, a housing market, housing and non-housing production sectors, a financial

intermediary sector, and a central bank. He solves this for optimal agent behavior and explores a number of policy questions. Jirnyi and Lepetyuk (2011) exactly solve a traditional Krusell and Smith Jr (1998) macroeconomic model with aggregate uncertainty, without needing to rely on abbreviations of agent information sets or of aggregate dynamics. While the author finds this approach impressive, I am interested in the additional dynamics which may be introduced by bounded rationality via learning. Yıldızoğlu et al. (2014) explicitly examines the same problem as Allen and Carroll (2001) and Howitt and Özak (2014), but Yıldızoğlu et al. (2014) use a neural network as part of their learning scheme and their agents successfully learn the optimal rule. This is an example of what might be called "artificial intelligence" learning, of which there is an extensive literature. Sargent (1993) outlines much of the early literature in this sub-field.

Recently, Gabaix (2014) developed a sparsity-based dynamic programming model which seeks to capture the idea that agents do not re-evaluate their behavior unless they are prompted by "big enough" events in their world. In his model these "big" events cause agents to re-optimize to find new behavior. This is somewhat similar to the "sticky expectations" of Carroll et al. (2011b) and Carroll et al. (2011a), wherein agents are slow to adjust the expectations they use to determine optimal behavior. These models have agents forming optimal behavior conditional on their current beliefs, where the learning-to-optimize behavior has agents acting non-optimally in nearly all periods, but in a way that they are moving in a direction of conditional optimality. To use an analogy, the learning-to-optimize behavior may be described as something like an intoxicated person stumbling home: they know where they want to go, and are heard roughly in that direction, but are stumbling to and fro on their way.

There is an older literature on "regret" in economics, which did not gain the foothold it might have. An early work in this literature is Loomes and Sugden (1982). This theoretical paper offers an alternate set of axioms for foundational choice theory, versus the traditional von Neumann-Morgenstern expected utility as described in Mas-Colell et al. (1995), based on comparing two outcomes for "regret" or "rejoicing" in a choice made. Loomes and Sugden

(1982) offer their theory as a simplification of the prospect theory of Kahneman and Tversky (1979). My current usage of regret is different than the one employed by Loomes and Sugden (1982). My usage of "regret" is a description of how agents might approximately attempt to maximize a von Neumann-Morgenstern expected utility objective function using a small sample of experience, while their regret-based theory sought to supplant von Neumann-Morgenstern expected utility entirely.

In the experimental literature, a number of authors have examined the role that learning plays in dynamic optimization problems. This includes Ballinger et al. (2003), Brown et al. (2009), and Carbone and Duffy (2014). Results of experiments have been mixed. In Chua and Camerer (2011), for example agents could find the optimal solution, but only after many "lifetimes." Ballinger et al. (2003) use multiple overlapping "generations" of subjects, to simulate what learning from predecessors may look like. They find that all their agents – including the best-performing third wave – do not get very close to learning the optimal rule. Brown et al. (2009) find that agents learn a near-optimal solution individually withing roughly four "life cycles," or roughly two when there is social learning.

Finally, Houser et al. (2004) sets up a difficult to solve intertemporal optimization problem and estimates the number of types of learners which appear in experimental laboratory data. They find three distinct and clearly identified types of learners, which they label "near rational," "fatalist," and "confused." This is a striking result, which sheds light on potential confounding factors in the previous experiments (different fractions of learning-types may not have been controlled) and motivates the search for agents which can learn a near-optimal rule from experience. This dissertation can be understood as examining one potential path by which the "near rational" learners found above may be modeled rigorously for quantitative macroeconomic and financial models.

1.6 Verification and Validation

All code will be open and published to a public repository. Verification ad validation are achieved in a number of ways. First, known results were replicated by each codebase. Specifically, the results of Allen and Carroll (2001) and Carroll (2012b) were each replicated a number of times independently. Second, each codebase has been independently constructed from scratch a minimum of two times and results compared against one another to confirm that they agree; all results have replicated appropriately. Finally, a number of simple "assert" tests have been used throughout the codebase to test code against known results. For example, when approximating a continuous distribution, test code asserts that the numerical mean agrees with the known analytical mean of the distribution. This has been repeated whenever possible.

1.7 Organization

The dissertation is arranged as follows: Chapter 2 introduces and explores regret learning, an individual-level model of learning to optimize. Chapter 3 introduces social learning and a novel relative-value estimator of the value of a consumption function. Chapter 4 introduces the Heterogeneous-Agent Computational toolKit (HACK) and the final chapter concludes.

Chapter 2: Regret Learning: Learning to Optimize from Individual Experience

2.1 Introduction

Economics has inherited an extensive toolset from applied mathematics. In their famous macroeconomic graduate text, Ljungqvist and Sargent (2012) title their introductory section "The Imperialism of Recursive Methods" – that is, models which can be described as dynamic stochastic optimization problems. The title is apt: nearly all theoretical economic models are centered around the optimization of a well-stated objective function, typically a discounted expected utility maximization problem, with a set of constraints and shocks such that the problem has a unique and tractable solution (either analytically, or more commonly, numerically tractable). The dynamic stochastic optimization problem is meant to represent the best ideal of economic agent behavior, and recursive methods are the prime solution method.

After adopting their methods, however, economics has not kept up with the mathematical dynamic programming literature. Beginning in the early and mid 1990s, the dynamic programming literature began to dabble in "learning-to-optimize" methods from the computer science literatures, searching for the underlying analytical reasons for those methods' success in practice. Bertsekas and Tsitsiklis (1996), "Neuro-Dynamic Programming," took a first pass at formalizing the relationship between dynamic programming methods and learning-to-optimize methods, and the dynamic programming literature exploded from that point forward with variations on that theme. The field is now extensive and fast-growing, and "approximate dynamic programming" and "reinforcement learning" – the terms from

applied mathematics and computer science, respectively – have firmly established themselves as the next wave of mathematical technology in solving large-scale dynamic optimization problems. One of the most popular texts, Powell (2007), "Approximate Dynamic Programming," is not subtle regarding the purpose of these methods. The full text title is, "Approximate Dynamic Programming: Solving the Curses of Dimensionality."[1]

The curse of dimensionality, of course, is the bane of the researcher using dynamic optimization methods. The curse goes like this: assume you want to represent a state space which has N dimensions. You represent this space with a grid with D points per dimension. The problem, of course, is that as N grows, the number of points in the space increase exponentially. If one has N=10 dimensions and D=5 points per dimension, this equates to $5^{10} = 9,765,625$ points. Now assume you must do an expensive calculation at each point – the time required to solve the entire problem can quickly becomes longer than the expected lifetime of our sun. This is one version of the curse of dimensionality. Powell (2007) outlines a number of different curses of dimensionality before happily explaining how the new methods can address many of them.

These new methods are true to both names mentioned above, "approximate dynamic programming" and "reinforcement learning:" they do, in fact, often approximate well known and well-understood dynamic programming methods such as value iteration or policy iteration. Portions of the traditional solution methods are approximated in ways which can be shown to still converge, or alternatively, are observed to converge frequently in practice. The other name, "reinforcement learning," is also indicative of the function of these methods: they are learning algorithms in the sense that they actively learn and take action while time is moving forward – there is no instantaneous attainment of the optimal behavior at time 0, which is then used for the rest of time. Rather the reinforcement learning algorithm pursues the optimal solution "online," while the model or real-life experience is underway. As the name suggests, the algorithms work by "reinforcing" values experiences. When constructed carefully, these methods reinforce their own behavior in such a way that it resembles the

[1] After publication in 2007, it has nearly 1600 citations on Google scholar as of the time of this writing.

solution methods in dynamic programming, only the reinforcement algorithms solve the problem by moving forward in time rather than backward as is often the case with dynamic programming.

These approximate dynamic programming and reinforcement learning methods largely function by using the same two pillars of computation that dynamic programming employs: the value function and the policy function.[2] Thus their methods solve the same basic problems that economists solve using the same core mathematical structure; only, the approximate dynamic programming approaches can solve much larger and more complex problems. One imagines that economists, largely taken with solving agent behavior as stochastic optimization problems, would welcome these developments.

It may be the case that the Great Moderation, the period of roughly the mid-1980s through around 2007 of low volatility in GDP growth and inflation, did not put much pressure on models and modelers to incorporate extensive detail and complexity in their models (see Bernanke (2004)). The abrupt end of the Great Moderation in the Great Recession put an end to that, however, and greatly raised awareness that, for example, representing heterogeneity in economic models is incredibly important (Carroll, 2012a). Or as Mankiw (2006) presciently observed, "There is nothing like a crisis to focus the mind."

Since the Financial Crisis and the Great Recession, there has been a flurry of activity and discussion about the vast differences between representative agent frameworks and other options. Heterogeneous-agent models, including agent-based modeling, which aims in particular to capture the details and complexity of the environment in which agents operate, has seen quite a bit of growth in the years since the Financial Crisis and Great Recession. See for example the special issues in the Journal of Economic Dynamics and Control: Anufriev and Branch (2009) and Assenza et al. (2013), or the invited talk Geanakoplos

[2]Assume that the agent solving a problem has a utility function or reward function which communicates how happy the agent is in any particular state. The value function is a form of "meta futility function" – it communicates to the agent how happy they can expect to be for the rest of their lives when they start in a particular state. It usually, but not always, contains discounting of future utility, and includes a measurement of the uncertainty the agent will face in the future. The policy function is essentially a behavior function: given a state, it tells what action an agent will take. The value function and policy function are related, as will be discussed further below.

et al. (2012) at the annual meeting of the American Economic Association. Heterogeneity and complexity are increasingly being recognized as important, but both features come at a cost – a complex model is difficult to solve for an optimal solution because of the curse of dimensionality, and some other alternative must be employed. What this alternative is has not yet been widely agreed upon. The discussion of the presentation Howitt (2013) at the Macro Financial Modeling Meeting in Spring 2013 is an insightful example. (See the reference in the Bibliography for excellent discussion.)

This again returns us to the question of why economists have not taken up the advances in ADP from the applied mathematics literatures. These methods address both the above concerns at the same time: ADP methods are explicitly intended to be used in complicated environments, where little may be known about the dynamics of the system or the stochastic process driving the system, *and* the methods are tied to an extensive, familiar, and widely accepted analytical framework for solving dynamic optimization problems.

The answer is likely multifaceted: first, as will be discussed in a moment, some economists *have* caught on, and there is a small but growing literature which is nipping around the edges of reinforcement learning and approximate dynamic programming methods. Second, there is a barrier to using much of the tools developed in the approximate dynamic programming (ADP) and reinforcement learning literatures. Because of the many lineages of ADP, the terminology and even notation can change significantly from one literature to another. Powell (2007) makes an explicit point of this difficulty in his introductory chapters, setting aside space specifically to work out a common-enough notation to communicate to multiple audiences. In addition, many of the solution methods are designed for problems which are less appealing for economists. For example Q-learning, one of the oldest and most popular methods of reinforcement learning, was designed explicitly for finite state and choice spaces, while many economic problems are stated in continuous terms. In addition, the convergence properties of Q-learning are often related to stochastic approximation techniques as in Robbins and Monro (1951) which are not popular in many economic literatures – the exception being least-squares macroeconomic learning, as described in Evans and Honkapohja (2001).

Agent-based computational economic (ACE) modelers are among the economists who have caught on to ADP techniques. For example, Sinitskaya and Tesfatsion (2014) use a variation on these methods for an agent-based macroeconomic model.Tesfatsion and Judd (2006) provides an excellent outline of early uses of reinforcement learning in economics. More recently, a handful of authors have explored reinforcement learning applied to intertemporal consumption-savings problems. Learning about intertemporal choice poses a particular challenge: obtaining enough information to estimate value and policy functions accurately requires many draws of time series. Since the time series are generated by agent experience, it can be quite difficult to obtain the number of draws needed. Lettau and Uhlig (1999) is one of the earliest such efforts; the authors use a classifier system (an early form of reinforcement learning) which chooses between an optimal rule and a version of the spendthrift ("consume everything") rule. Their model learns the non-optimal rule under a wide range of conditions, and the authors caution readers that households in practice may not learn optimal rules for similar reasons. Interestingly, Başçı and Orhan (2000) revisit the Lettau and Uhlig (1999) results and find that agents are once again able to distinguish the optimal rule from non-optimal rules when "trembling hand" experimentation is allowed. Allen and Carroll (2001) use a clever parametrization of the consumption function for a simple consumption-savings problem and employ a first visit Monte Carlo reinforcement learning estimator[3] to chose the best policy on a fixed grid of policies. They find that (a) agents can find a near-optimal rule given enough time, but (b) enough time can be millions of periods, rendering the learning not useful for practical time parameterizations. They suggest that one solution to this may be social learning, which is explored in Chapter 3. Howitt and Özak (2014) and Özak (2014) address the same consumption problem as Allen and Carroll (2001), but using a clever form of policy gradient learning, a form of reinforcement learning[4] which utilizes marginal utilities to update a consumption function.

[3]The two basic paradigms for reinforcement learning use either Monte Carlo estimators or Temporal Difference estimators, or some combination of the two. See Sutton and Barto (1998), Chapter 5, for an intuitive overview of Monte Carlo reinforcement learning.

[4]See Sutton and Barto (1998) for an intuitive overview.

They obtain agents who can find a near-optimal rule quickly but must know more about the structure of the problem than the Allen and Carroll (2001) agents.

An alternative application of reinforcement learning is finding optimal solutions when traditional techniques are intractable. Since I am interested in reinforcement learning as a description of bounded rationality, I will not explore this usage extensively in my own work. Examples of this include Hull (2012) and Jirnyi and Lepetyuk (2011). Hull (2012) constructs an overlapping generation model with 60 rolling cohorts of agents, a housing market, housing and non-housing production sectors, a financial intermediary sector, and a central bank. He solves this for optimal agent behavior and explores a number of policy questions. Jirnyi and Lepetyuk (2011) exactly solve a traditional Krusell and Smith Jr (1998) macroeconomic model with aggregate uncertainty, without needing to rely on abbreviations of agent information sets or of aggregate dynamics. While the author finds this approach impressive, we are interested in the additional dynamics which may be introduced by bounded rationality via learning. Yıldızoğlu et al. (2014) explicitly examines the same problem as Allen and Carroll (2001) and Howitt and Özak (2014). Yıldızoğlu et al. (2014) use a neural network as part of their learning scheme and their agents successfully learn the optimal rule. This is an example of what might be called "artificial intelligence" learning, which will not be discussed in great depth.

2.1.1 Regret Learning

The dynamic stochastic learning algorithms noted above required either extensive knowledge of the problem to be solved, high memory, or both. Regret learning is a low-cost dynamic stochastic learning method which requires relatively minimal memory and knowledge of the problem structure. As the name indicates, the central idea is "regret" on the agent's part – having gained some experience, the agent looks back and says, "if I knew then what I know now, I would have acted differently..." and then uses how he/she should have acted differently to inform future action.

The key is that the agent is learning a very rough value function from experience (as

in Allen and Carroll (2001) and Chapter 3). Value functions for dynamic consumption-savings problems are typically continuous functions of the state space. When one solves for the optimal value function in dynamic programming, interpolation over a grid on the state space is used to capture the fact that the true function is continuous.

Regret-learning agents, however, perceive the value of their states one period ahead in coarse terms: they partition the state space into a set of contiguous "bins" such that they view the value of being in a bin the following period as the same for all states in that bin. Agents are capable of understanding that their current state will influence how conditionally likely they are to land in one of the state partitions the following period; this knowledge is entirely gathered from experience. The coarse partitioning of next-period states attempts to capture the idea that agents in practice don't think about an infinite number of plans tomorrow, but rather make plans based on a handful of broad possible states of the world – for example, "good", "average," or "bad," or a similar categorization.

An agent starts with an arbitrary consumption function and arbitrary partition of the state space for forming the next-period conditional value function. For my purposes here all agents will begin with the spendthrift ("consume everything") consumption function.[5] Agents then experience random income shocks and simply follow their consumption rule for D periods. Agents take a weighted average of their experience and this produces the "value bins." After D periods, the agent has new knowledge he/she didn't know before, in terms of the value estimates. The agent has also learned a little about the income process.

The rough knowledge of the value function at a few points coupled with the observed income process and knowledge of the law of motion of one's own state variables allows the agent to look back and say, "if I knew then what I know now, I would have made different choices..." Call these the agent's "regret choices." The agent then chooses a new consumption function which comes as close as possible to these "regret" choices – this provides a new consumption function to be used for the next episode in the agent's life. The agent can repeat this process indefinitely – use a function for a while, pause and learn

[5]The spendthrift consumption rule is intuitively appealing and simple to implement, as well as close to a "worst case" consumption function in a world where agents cannot go into debt.

a new consumption function from mistakes, repeated until the end of life.

To summarize, regret learning is a two-step learning process: in the "value estimation" step, agents first estimate the value function associated with a particular consumption function using their experience. In the "policy improvement" step, agents then use the estimated value function to improve their consumption function. The policy function which emerges produces the best choices (in a minimized squared errors sense) *conditional on the values learned thus far from experience*. A key insight is that restricting agents to only the value estimation step of this process gives the appearance of *significantly slow learning*, because value estimation from limited experience is an extremely slow process. However when the agent can learn an improved policy from even a poorly estimated value functions, the agent can leap through the policy space quickly. This is particularly true when the value function estimate maintains specific qualities, namely the when it maintains the curvature related to the true value function. Thus agents can make exceptional progress in improving their consumption choices even in a very noisy world with poor estimates of the true value of each state. Importantly, the current formulation of agent learning does *not* protect agents from learning a "bad" value function from a stream of not-representative experiences. I explore this feature of the model in more detail in Section (2.5) below.

2.1.2 Preview of Results

The number of partitions an agent uses to value next-period states is a simple and intuitive lever by which I can change the sophistication of an agent. An agent with three or five partitions, for example, may be categorized as less sophisticated as an agent with, say, 25 partitions. A sophisticated agent, with 25 or 55 bins, for example, can learn a good value function rather quickly. However even a not terribly sophisticated agent, with perhaps 3 or 5 bins, may learn a passable policy rather quickly.

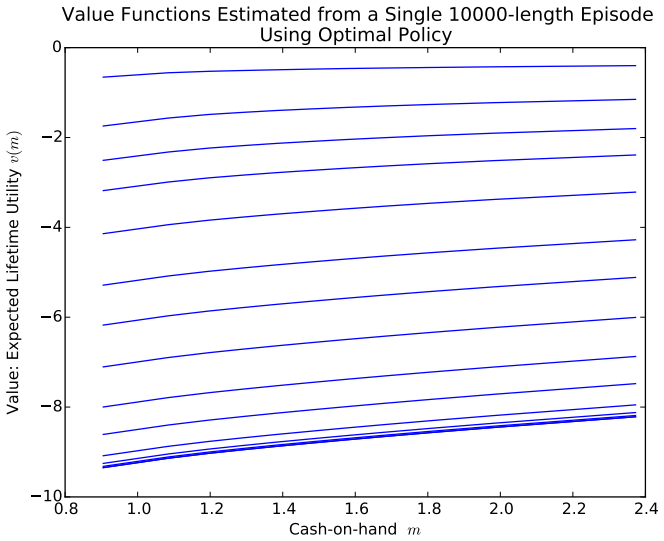
Regardless, both sophisticated and non-sophisticated agents are prone to "learn the wrong thing" if they get a series of particularly good or particularly bad shocks. Agents can still re-learn a good policy when the next set of shocks is more representative of the

underlying distribution; this characteristic is a feature rather than a bug when explaining human behavior in practice.

All of this occurs without the regret learning agents knowing anything about their income process except their own experience. As discussed below, all features of the agent's solution-by-learning are constructed from agent experience. This is key for the learning process to the usable in large-scale agent-based models. If the researcher must inform the agent about extensive details about the nature of the optimization solution, the hydra of computational intractability quickly raises its multi-dimensional heads.

Figures (2.1) and (2.2) displays the convergence of a value function estimate for an agent who has been handed the optimal consumption function and is only estimating the value of this function from experience (not attempting to improve the policy). The purpose of these figures is to demonstrate the difficulty of estimating value functions from experience, even when the agent has been given the true optimal policy, a moderate sophistication level (in terms of number of partitions of the state space) of 11 bins, and an extensive amount of time to learn learn, 10,000 periods. Figure (2.1) shows that learning a value function from experience does converge in a traditional-looking sense. However figure (2.2) demonstrates that even under high sophistication and extensive learning time, this function has not converged to the true value function but is rather a biased function offset from the true function. (The two estimates displayed are for two alternative ways of sampling from the shock space – either using a true continuous random sample drawn from the continuous distribution, or a sample drawn from a discretized version of the shocks space.)

The bias, however, is only for a single realization of learning from experience. Figure (2.3) shows that the value functions learned by many agents are still correct in the aggregate. When I average the value function estimates from a large number of agents learning the value function from experience, we can see that the mean value function is an extremely close match to the true value function. Figure (2.4) highlights this be stripping out the distribution of functions and only displaying the mean value function. The difference in utility terms between this value function and the true value function is trivial and nearly



Estimates shown for periods 1,3,5,7,10,15,20,27,37,50,70, 90, 115, 400, and 10,000.

Figure 2.1: Learning the value of the optimal policy c^* using 11 bins and 10,000-period experience

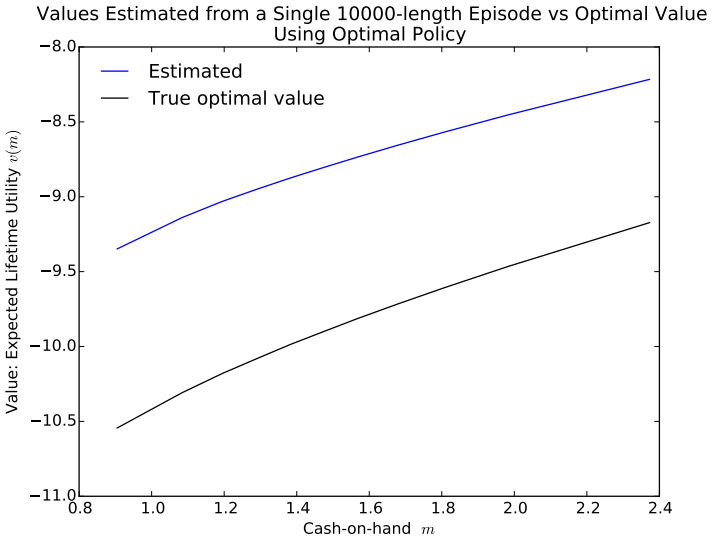


Figure 2.2: Bias in learned optimal value function $\hat{v}^*(.)$ using 11 bins and 10,000-period experience

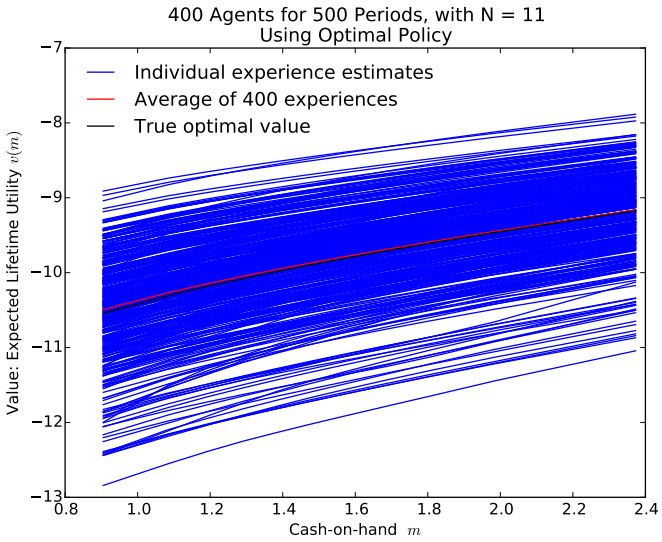


Figure 2.3: Average $\hat{v}^*(.)$ results over 400 agents, using 11 bins for 500 periods

nonexistent.[6] Agents are capable of learning about the true value function from experience, but only in a distributional sense – this idea is the motivation behind the social learning scheme discussed in Chapter 3, but I will not explore social learning further in this chapter except as an extended application of regret learning.

Figures (2.1), (2.2), (2.3) and (2.4) illustrate how much time is required to get a high-quality estimate of the value function purely from experience. As noted above, however, an agent can learn a policy function from even a poorly estimated value function. I leave the details of how an agent creates this policy function until the methods sections; suffice it to say that the agent estimates a value function following D periods of experience (where this is a parameter I can vary) and then looks back, decides which consumption choices *would* have optimized the value function they just learned (which is perhaps very poorly estimated in absolute terms) at each state they experienced – call these the "regret choices," and then forms a new policy function which minimizes the squared errors between the new policy

[6]Note that this precision arises from 400 agents learning for 500 periods, each with 11 bins partitioning the next-period state-space. These numbers were chosen because the accuracy of the mean value function improves only marginally more as either number of agents, number of periods, or number of partitions is raised further.

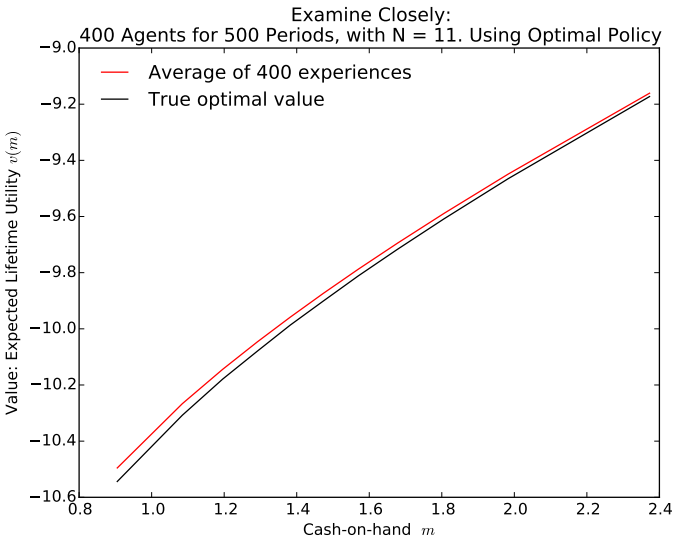


Figure 2.4: Average $\hat{v}^*(.)$ results over 400 agents, using 11 bins for 500 periods examined closely

function and the regret choices. Then the process is repeated.

Figures (2.5) and (2.6) demonstrate what the first step in this process looks like for an agent with 11 bins and $D = 6$ and $D = 20$ periods of learning, respectively. The red dashed line represents the true optimal solution to this consumption problem, the stars represent the "regret choices" of the agent, and the solid blue line is the new consumption function the agent has just learned from their "regret" choices. The vertical dotted lines indicate the 5^{th} and 95^{th} percentile cutoffs of the state variable's ergodic distribution under the optimal rule – this gives us a rough estimate of how far this particular set of "regret values" falls from the those under the optimal distribution. It will be demonstrated that when experiences are particularly high or low with respect to the ergodic distribution of the state variable, the agent is more likely to learn a less-optimal consumption function. That is, as one would expect, if an agent is learning from experience, when the experience is highly unusual the agent learns to "make mistakes" from the perspective of the rational solution – however for the agent, the "incorrect" policy function simply represents the conditional-on-experience, approximately rational, improved consumption function.

Figure 2.5: Learning an Improved Policy From Regret: 11 bins, $D = 6$ learning periods, $K = 1^{st}$ Learning Episode

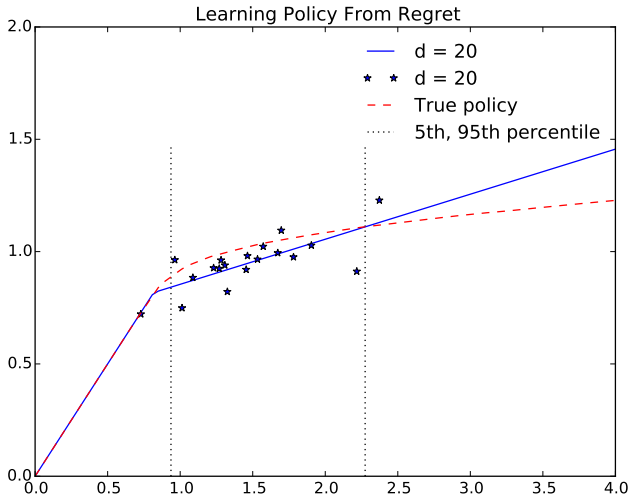


Figure 2.6: Learning an Improved Policy From Regret: 11 bins, $D = 20$ learning periods, $K = 1^{st}$ Learning Episode

Finally, note that the learned consumption function in figures (2.5) and (2.6) are the agent's *first* learning steps for each value of D – the agent started with the spendthrift "consume everything" policy function at period zero, and these are their first improved functions. It can be seen that this "sophisticated" agent gets close to the optimal policy function quite quickly. Agents will depart from the optimal policy function, however, for at least two reasons: because the agent is "less sophisticated" (i.e. has less bins), or as noted above, because the agent experiences a set of states which are low probability with respect to those under the true optimal rule.

I now turn to a more formal discussion of regret learning. The following sections first outline the problem the consumer desires to solve optimally (Section 2.2) before discussing one class of solution method for solving the optimization problem "instantaneously" from the consumer perspective (Section 2.3). With those preliminaries in place, Regret Learning is introduced and described in detail in Section (2.4), and important theoretical and practical questions are raised concerning its performance. These questions are immediately answered in the Results, Section (2.5). Finally, Section (2.6) concludes with a summary and a description of many next steps in this research program.

2.2 The Consumer Problem

The learning-to-optimize behavior I discuss will be explicitly applied to a stationary, infinite-horizon dynamic optimization problem. Appendix C outlines an extended finite-horizon problem which can be transformed into the infinite-horizon problem by adding a simple Poisson probability of death each period, providing a simple justification for using an infinite-horizon problem as a learning target. Additionally, both Gourinchas and Parker (2002) and Cagetti (2003) note that households in the data appear to act as though they are solving infinite-horizon problems until around age 45-55, at which point they apparently realize that retirement is fast approaching and begin to save as though they are solving a finite-horizon problem as in Appendix C).

2.2.1 Infinite Horizon Consumer Problem

The basic household consumption-savings problem under uncertainty can be stated as follows. Households have an uncertain income and may either save money at a risk-free rate of return or spend it immediately on consumption. Consumption produces rewards for the household via a utility function; this utility function is used to create a lifetime objective function. Savings produce rewards only in so far as they represent the ability of the household to eventually consume in the future.

The objective is to maximize total expected lifetime utility. The household problem, then, can be stated as follows:

$$\max_{\{c_t\}_{t=0}^{\infty}} \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right] \tag{2.1}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1} \text{ the law of motion,}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \; given.$$

Where:

- y_t is a random income shock each period, normalized and distributed as an IID lognormal random variable with mean 1 and $\sigma_y = 0.2$,
- c_t is consumption in period t,
- m_t is "cash-on-hand," the total **m**onetary resources under control of the household after interest accrues on savings and wages y_t are paid,

- $a_t \equiv (m_t - c_t)$ is savings,
- $R = 1.03$ is deterministic, risk-free return on savings every period,
- $\beta = 0.95$ is the discount factor, and
- $u(.)$ is a Constant Relative Risk Aversion (CRRA) utility function with risk aversion $\rho = 3$.

The calibrated parameter values are summarized in Table (2.1). Utility takes the CRRA form:

$$u(c) = \frac{c^(1-\rho)}{1-\rho}.$$

Carroll (1997) and Carroll (2001b) provide excellent background for the usage of this functional form for this problem (and related difficulties).

The solution to the discounted dynamic optimization problem (C.6) is the infinite sequence of consumption choices $\{c_t\}_{t=0}^{\infty}$ which maximizes the problem's expected discounted utility stream. Regularity conditions on both $u(.)$ and the law of motion $m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$ guarantee that a solution exists for this problem, and in addition, the optimal consumption vector $\{c_t^*\}_{t=0}^{\infty}$ can be produced by an optimal policy (consumption) function $c_t^* = c^*(m_t)$ which is a function of the state space. Thus the apparently intractable problem of choosing an infinite-length vector becomes a simpler problem of choosing a function which maps an infinite stream of states provided by the law of motion into a stream of consumption choices.

Table 2.1: Parameters and Sources

Parameter	Value	Description	Source
β	0.95	Geometric discount factor	Allen and Carroll (2001)
ρ	3.0	Risk aversion	Allen and Carroll (2001)
σ_y	0.2	Shock to income	Carroll (1992)
R	1.03	Return factor, savings	U.S. Treasury returns 1990s

If I denote the optimal consumption policy function c^* then the optimal value function $v^*(m)$ is simply the value of starting in state $m = m_0$ in the expected discounted infinite sum in problem (C.6):

$$v^*(m) = \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c^*(m_t)) \right] \tag{2.2}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$$

$$c_t, m_t \geq 0; \;\; m = m_0 \; given.$$

Note that the above relationship holds not only for the optimal consumption function c^*, but also any arbitrary consumption function such that the appropriate Blackwell Sufficiency Conditions hold for the problem.[7] Let an arbitrary consumption function be parameterized by a set of parameters, which I will collectively denote θ, and define such a consumption function as c^θ. Then under the appropriate conditions, the associated value function can be derived from c^θ as follows:

$$v^\theta(m_0) = \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c^\theta(m_t)) \right] \tag{2.3}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$$

$$c_t, m_t \geq 0; \;\; m_0 \; given.$$

[7] See Appendix A for a full discussion of the requirements on the structure of the problem for there to exist a unique fixed-point value and policy function solution. Unless otherwise noted, the assumptions will be made for all proceeding discussion.

As discussed below, given an arbitrary value function v^{arb_1}, I can derive an associated policy function c^{arb_1}, and given an arbitrary policy function c^{arb_1} I can derive an associated value function v^{arb_2}. However this process will *not* produce $v^{arb_1} = v^{arb_2}$ unless I have discovered the optimal policy and value functions.[8] This property is key for the policy iteration dynamic programming solution and will provide key intuition for the regret learning algorithm to be defined below.

When the solution as described above exists, the problem can be re-written in Bellman form:

$$v(m_t) = \max_{c_t} u(c_t) + \beta \mathbb{E}_t \left[v(m_{t+1}) \right] \tag{2.4}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \; given.$$

The following section will discuss when the solution to this problem exists. (See Appendix A, Carroll (2012c) for a lengthy discussion of when the solution for this problem exists.)

This form breaks down the consumer problem into a much simpler form: the agent is simply trading off between the utility of consumption today, denoted by $u(c_t)$, and the expected future lifetime utility of the consequence of the choice today, denoted by $\beta \mathbb{E}_t \left[v(m_{t+1} \right]$.

[8] Or rather, $|v^{arb_1} - v^{arb_2}| \leq \delta$ for some tolerance δ.

2.2.2 Buffer Stock Solution Form

I have discussed an approximate policy function c^{θ}; now we outline the specific parametric form of this approximation. Under mild conditions and the parameterization of this problem,[9] the optimal consumption function takes the form:

$$c^*(m_t) = E[y_t] + g(m_t - \bar{m}^*),$$

where g is a nonlinear function resulting from optimizing problem (C.6) and $\bar{m}^*$ is the target buffer stock savings level for liquid wealth which exists under my conditions. I abuse notation slightly by denoting both the consumption in period t as a function of the cash-on-hand state variable m_t: $c_t = c^*(m_t)$ in the case of the optimal consumption function, or $c_t = c^{\theta}(m_t)$ in the case of an arbitrary approximate consumption function parameterized by θ, to be described further below.

When a consumer experiences a shock which pushes m_t away from $\bar{m}^*$ he/she will consume such that in expectation next period, $E[m_{t+1}]$ will move towards $\bar{m}^*$. As noted in Carroll (2012c), the exact form of the function g is highly nonlinear and difficult to describe analytically. Allen and Carroll (2001) propose a simple piecewise linear approximation to this function which has extremely low welfare cost and an intuitive parameterization, which will be used for the learning algorithm outlined in this paper. A more extensive discussion of the properties of this approximation can be found in Allen and Carroll (2001), Özak (2014), and Chapter 3. A first-order Taylor approximation taken around $\bar{m}^*$ gives us a linear consumption function with an intuitive interpretation:

$$\widetilde{c}^{\theta}(m_t) = E[y_t] + \kappa\left[m_t - \bar{m}\right], \tag{2.5}$$

[9] The condition in this version of the model is $\mathsf{R}\beta < 1$. Intuitively, this is a statement about the "patience" level of the consumer versus potential growth in savings. See Carroll and Samwick (1997) and Carroll (2012c) for detailed discussion regarding this condition.

where $\theta \equiv (\kappa, \bar{m})$ and the tilde "$\widetilde{c}$" denotes that this is an intermediate step to the final form c^{θ}. κ is the marginal propensity to consume out of wealth[10], and $\bar{m}$ has the same interpretation as before as the buffer-stock savings target. To obtain the final piecewise linear consumption-function form, I simply impose the liquidity constraint as follows:

$$c^{\theta}(m_t) = \begin{cases} m_t & if\ \widetilde{c}^{\theta}(m_t) \geq m_t \\ \widetilde{c}^{\theta}(m_t) & otherwise. \end{cases} \tag{2.6}$$

This complete restriction on borrowing is imposed here for simplicity of exposition. However note that the imposition of the borrowing constraint is not as restrictive as it may first seem. Carroll et al. (1992) outlines evidence for consumers facing a low but non-zero probability of a zero-income shock occurring at the annual frequency, which differs from the distribution of shocks typically faced by the consumer. Any rational consumer who faces such a process would self-impose a borrowing constraint each period alive and with a positive probability of a zero-income even each period this collapses to a complete liquidity constraint used here. Even if the consumer did not rationally self-impose this constraint, a rational lender may do so. Regardless, a straightforward extension is to implement a borrowing constraint which is linear in m_t.

2.2.3 Welfare Cost of Approximate Solutions

As noted in the discussion following expression (2.3), for an approximate policy function c^{θ} I can obtain an associated value function v^{θ}. If I have also solved for the optimal value function v^{*} for this problem, I can calculate a measure of welfare cost implied by following the approximate policy function c^{θ}. Following Allen and Carroll (2001), I call this value a "sacrifice value." For a consumer following the optimal consumption rule, the sacrifice value represents the maximum amount the consumer would be willing to pay to to avoid

[10] To see this, take the derivative of $c^{\theta}(m_t)$ with respect to m_t.

switching permanently to the non-optimal rule. For a parameter θ, denote this value ϵ^{θ} and derive it as follows:

$$v^{*}(m-\epsilon^{\theta}) = v^{\theta}(m) \;\; \forall_m$$

$$\Leftrightarrow m-\epsilon^{\theta} = v^{*\,-1}(v^{\theta}(m)) \;\; \forall_m$$

$$\Rightarrow \epsilon^{\theta}(m) = m - v^{*\,-1}(v^{\theta}(m)) \;\; \forall_m,$$

and, using the ergodic distribution F_m of m under the optimal policy c^{*}[11], calculate the expected sacrifice value as:

$$\bar{\epsilon}^{\theta} = \int_{\bar{q}}^{\infty} \epsilon^{\theta} dF_m.$$

I use $\bar{\epsilon}^{\theta}$ to identify the expected sacrifice value for any given approximate rule c^{θ}. This provides an explicit way to compare non-optimal rules with the optimal solution, allowing us to map the effectiveness of learning and rules directly to more traditional methodology.

Since $\bar{\epsilon}^{\theta}$ can be calculated for any approximate function c^{θ}, it is simple to construct the surface of welfare costs over any given range of consumption function parameters. Figure (2.7) displays the contour plot of this surface for $\bar{m} \in [0,\ 3]$ along the y-axis and $\kappa \in [0,\ 1]$ along the x-axis. This should be read like a contour map – each line denotes approximate consumption rules which have equal sacrifice values. A few things can be quickly observed. The c^{θ} function with minimal sacrifice value – that is, the c^{θ} which is closest to c^{*} in utility terms – occurs at $\bar{m} \approx 1.4$, $\kappa \approx 0.15$ and has a sacrifice value of $\bar{\epsilon}^{\theta} = 0.0055$. I will denote the best approximate consumption function $c^{\theta *}$. That is slightly more than one half of one percent of expected annual income, very close to the true optimal rule. Since each line has

[11] See Carroll (2001a) for a discussion of the ergodic distribution of cash-on-hand following an optimal buffer-stock rule, as well as the methodology used to generate the ergodic distribution.

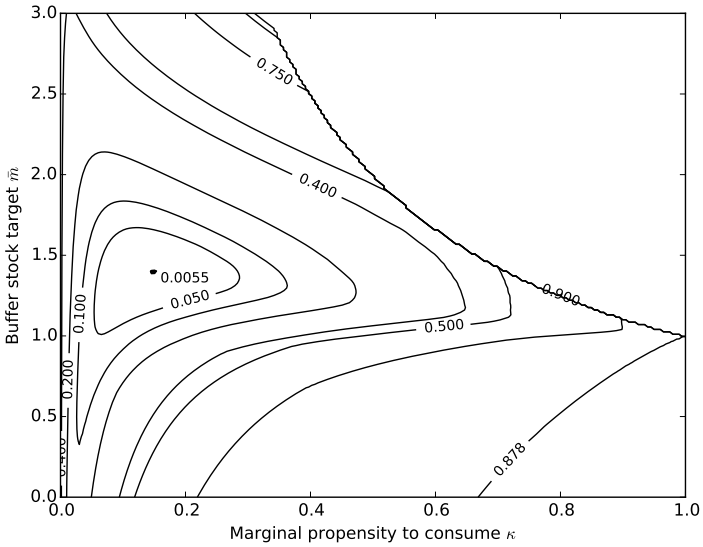


Figure 2.7: Sacrifice Values for Approximate Consumption Functions

the associated sacrifice value value printed on it, the inner-most ring around the minimal sacrifice point indicates a sacrifice value of 0.05, or 5 percent of expected annual income. The point which represents the "consume everything" consumption rule is at (1.0, 1.0); the sacrifice contour which intersects here has a value of 0.878.

The large "empty" region in the upper right-hand portion of Figure (2.7) is due to the fact that the parameters in this area produce consumption functions which tell a consumer to consume zero for a non-trivial number of m-values. For example, Figure (2.8) displays one such consumption function, with $\bar{m} = 2.5$ and $\kappa = 0.8$. It is clear that for some values of cash-on-hand, about $m \leq 1.25$, the consumption function tells the consumer to eat nothing. This is a particular problem for the consumer, because the CRRA utility function used here goes to infinity as consumption goes to zero: $u(c) \to \infty$ as $c \to 0$. Thus the boundary of the empty region in Figure (2.7) is simply those rules which would force an agent to consume zero over a non-trivial portion of the state space.

The reason for the small welfare cost of the best approximate consumption rule $c^{\theta *}$ can be quickly seen when examining the function against the true nonlinear optimal rule c^*.

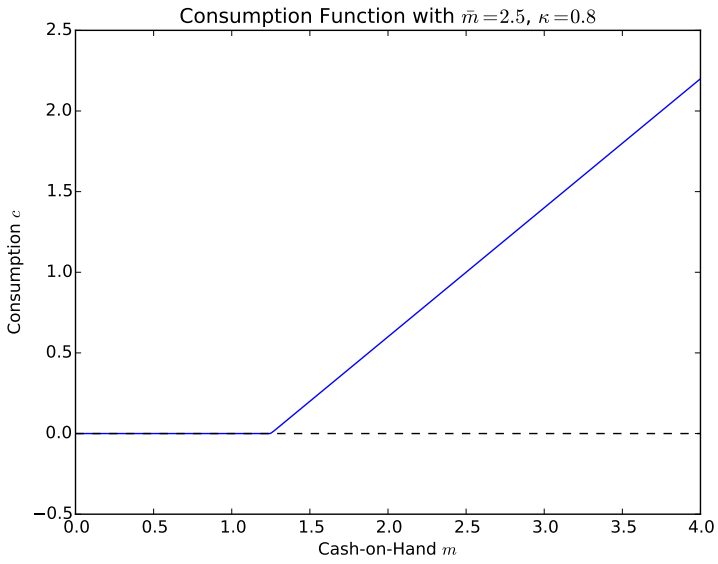


Figure 2.8: Example of a "Bad" Consumption Function

These are both displayed in Figure (2.9). Also displayed are the 5^{th} and 95^{th} cutoffs for the ergodic distribution of cash-on-hand m when following the optimal policy c^*, denoted F_m. Between the 5^{th} and 95^{th} percentiles, the best approximate policy is extremely close to the true optimal policy. Thus, for about 90% of the time, an agent using the approximate optimal rule $c^{\theta *}$ will be extremely close to the true optimal rule in welfare terms.

There is actually a deeper question at hand when one asks the welfare cost of a learning algorithm. The sacrifice value noted above is a value that assumes an agent follows the non-optimal rule for the rest of their lives – this is what the value function represents, and the value functions are used to calculate the sacrifice value. This is an important and reasonable first pass. However in learning agents are likely not using the same rule for the rest of their lives. The value function associated with the *entire learning process* is almost certainly different form that of following a specific rule. This "meta" value function of learning may not even be monotone, which is a requirement for measuring unique sacrifice values. At the intuitive level, if it takes a non-trivial set of periods in a finite life to try another policy, the opportunity cost of agent time is the best policy the agent has seen so

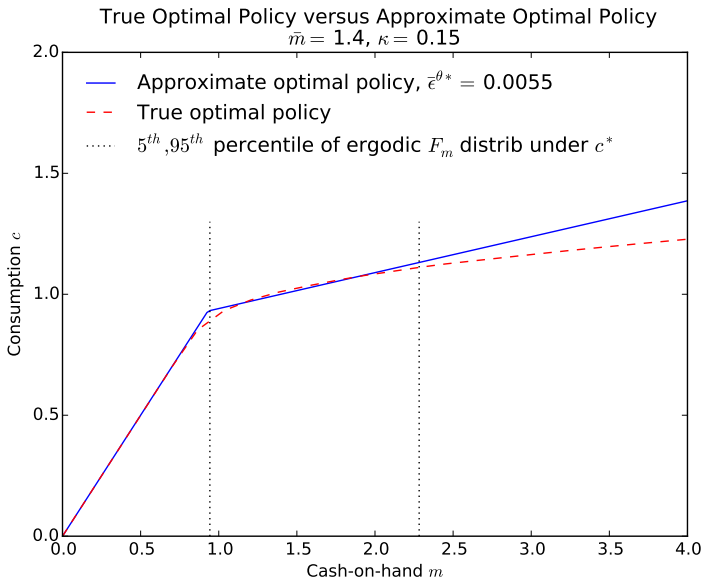


Figure 2.9: True vs Approximate Optimal Consumption Functions

far – presumably the current one. How does the agent decide when experimenting with a new policy is not worth it? There's a risk aversion decision buried in this decision which has not been explored extensively in the economic literature, in part because it requires agents to be using non-optimal rules to guide their behavior.

This is a question has been understood for a long time in the reinforcement learning literature, to be discussed below. In that literature this question is known as the "exploitation/exploration trade-off," and is most cleanly illustrated by a set of simple problems called "bandit" problems. See Sutton and Barto (1998) and Auer et al. (2002) for an excellent description of both the exploration/exploitation trade-off and its illustration in bandit problems. I do not deal with this deeper question of the welfare costs of learning here, instead leaving this to future work. I will simply note that the exploration / exploitation trade-off is a key fundamental question as soon as one moves away from instantaneous optimization. I strongly suspect that many behaviors, such as satisficing, have a basis in this trade-off.

2.3 Policy Iteration Solution Method

Economic agents generally function under what might be called "instantaneous optimization" – in any period, the optimal solution (2.2) is obtained instantaneously and handed to the agent, who then proceeds to enact this optimal behavior. Note that solving for this optimal behavior may be very computationally intensive, particularly in general equilibrium with heterogeneous agents. In this case, the true distribution for expectations may be a very high-dimensional object. Even aside from behavioral evidence that cast human abilities to solve complex dynamic optimization problems into doubt,[12] the requirement that agents solve a program such as (2.2) optimally and rationally[13] imposes extreme computational burden on the agent and thus the researcher as a model becomes more heterogeneous (in terms of agent attributes and behavior) and complex (in terms of both model structure and dynamical properties). Large-scale models such as Geanakoplos et al. (2012) are intractable if rational optimizing behavior is imposed on agents. None-the-less, the opposite alternative – agents who have completely fixed behavioral rules, which do not change as their world changes – is also unattractive. Such a solution immediately re-raises the specter of the Lucas (1976) Critique, ready to be restated for a new generation:

> "This essay has been devoted to an exposition and elaboration of a single syllogism: given that the structure of an econometric model consists of optimal decision rules of economic agents, and that optimal decision rules vary systematically with changes in the structure of series relevant to the decision maker, it follows that any change in policy will systematically alter the structure of econometric models."

Even if the argument is made that agents should *not* be modeled as optimizing decision-makers, and thus the structure of econometric models need not be bound to this, it is

[12] See Houser et al. (2004) study which clearly outlines a spread in human ability to solve dynamic programs optimally.

[13] That is, the expectations under which the agent solves (2.2) are the true and correct mathematical expectations for the environment in which the agent functions.

hard to argue that all agents should have fixed rules which do not adapt to changing environments. In his excellent "Micromotives and Macrobehavior," Schelling (2006) notes both the importance of modeling purposive problem-solving as well as the difficulty of capturing this in practice:

> "With people ... we usually believe we are dealing with conscious decisions or adaptations in the pursuit of goals, immediate or remote, within the limits of their information and their comprehension of how to navigate through their environment towards whatever their objectives are. In fact, we can often ascribe to people some capacity to solve problems – to calculate or to perceive intuitively how to get from here to there. ... [However] we can get carried away with our image of goal seeking and problem solving. We can forget that people pursue misguided goals or don't know their goals, and that they enjoy or suffer subconscious processes that deceive them about their goals. And we can exaggerate how much good is accomplished when people achieve the goals we think they think they have been pursuing."

Like Allen and Carroll (2001), Howitt and Özak (2014), and others mentioned in the introduction, I propose that *learning-to-optimize* is a reasonable middle ground between agents who fully optimize and agents who have fixed behavioral rules. Similar to Allen and Carroll (2001) my regret-learning agents estimate the value of a current approximate policy c^θ by forming a noisy estimate of the discounted sum in expression (2.3). In Allen and Carroll (2001), agents took a coarse discretization of the state space, identified their "bin" in this discretization, and then formed an estimate of the value function v^θ in equation (2.3).[14] By necessity, these agents only estimate a portion of the value function v^θ; namely the conditional value of starting experience in their "bin" in the state space. Under strict assumptions – for example, agents must always "restart" each draw of their experience in the same "bin" as their initial draw – the portion of the value function estimated by

[14] See Appendix B for a more extensive discussion of the solution method of Allen and Carroll (2001).

an agent can be shown to converge to the equivalent portion of the true value function. Chapter 3 exploits this structure to allow agents to socially learn about consumption rules from "similar types" of agents – that is, agents who began their estimation process in the same partition of the state space as themselves.

While agent learning in Allen and Carroll (2001) and Chapter 3 can be shown to converge to the point-estimate of the value function associated with each bin, each agent individually knows nothing about the rest of the value function and thus is restricted in two important ways: first, agents cannot learn from the broader members of the population who are of different "bin types," and second, agents cannot directly learn about potentially better choices from their *own* value function estimate because it only covers a small portion of the state space. To explore the parameter space governing their consumption function they must effectively grope blindly, or talk to other agents who are potentially also groping blindly through the policy space.

Regret learning seeks to alleviate both these points via a simple extension of the Monte Carlo-style value function estimator in Allen and Carroll (2001) and Chapter 3. This extension allows agents to learn about the entirety of the value function across the state space from their own experience. They learn a biased version of their value function, but none-the-less this value function provides them enough information to learn from their own mistakes in a such a way as to quickly settle into a stable and well-defined space around the optimal solution, as demonstrated by numerical results below.

Furthermore, the extension is conducted in such a way as to create a "convergence rich" environment. As will be discussed in more detail below, each component of the learning program has been constructed such that individually it will converge to its appropriate target were it to function alone. Convergence has not yet been proved for the system as a whole, but the simulation results to be described below are extremely promising in this regard and it is only a matter of time before the full analytical description is complete.[15]

[15]Singh and Sutton (1996) prove that a similar idea, "every-visit" Monte Carlo value function estimation, is a biased estimator in finite experience, with bias going to zero as experience goes to infinity. The regret-learning estimator is biased in a similar was as the every-visit estimator. In particular see the setup and proofs in Section 3.3 of Singh and Sutton (1996). The perpetual-youth Poisson probability of death employed

2.3.1 An Intuitive Description

Consider the agent problem presented in expression (C.6), restated here for convenience:

$$\underset{\{c_t\}_{t=0}^{\infty}}{max} \; \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right]$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1} \;\text{ the law of motion,}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \; given.$$

An agent i begins with an initial policy $c^{\theta_{k=0}}$. Although I will refer to agent i, all subscripts denoting agent i are dropped from the following discussion to reduce notational clutter. The agent experiences an indefinitely long income stream $\vec{y}$, subdivided into D-length episodes, each episode denoted by the subscript k. For example, the first $k = 1$ episode would be denoted $\vec{y}_k \equiv [y_0, y_1, ..., y_t, ..., y_{D-1}]$. For the duration of the episode, the agent simply follows the current consumption policy and records a rolling sum of discounted utility of the consumption experienced. At the end of the D-length episode k, the agent has formed a rough approximation, $\hat{v}^{\theta_k}$, to the true value function v^{θ} associated with c^{θ}.

Using this newly generated value function $\hat{v}^{\theta_k}$, which the agent did not have at the beginning of the episode, the agent can look back on the shocks experienced and ask, "if I knew then what I know now" – namely the $\hat{v}^{\theta_k}$ function – "what optimizing consumption choices *should* I have made?" This provides the agent with a set of choices that *would*

to motivate the infinite horizon version of the problem allows regret learning to fit the abstract two-state Markov chain framework Singh and Sutton (1996) employ, and the parallel application of the update across states in regret learning somewhat mirrors the every-visit state update. The similarities are not enough for a direct application of their proof, but points to one possible direction for analytical results.

have been optimal under the $\hat{v}^{\theta_k}$ value function. Call these consumption choices the "regret choices" – what the agent *should* have done, if they'd only known what they know now. The agent then finds the consumption function of the form in equation (2.6) which is closest to the regret choices in a sum of squared errors sense. This produces the next consumption function to employ, $c^{\theta_{k+1}}$, and the process is repeated.

Thus in a specific, technical sense, the agent is using "regret" – the act of looking back on past choices and determining what they should have done given knowledge learned from experience – to ascertain the next best course of action. As will be described in the Section (2.5), this can provide agents with a relatively rapid convergence in expectation to a well-defined neighborhood around the optimal solution.

I will use policy iteration as described below as a framework for regret learning. I will first set up the analytical framework of the policy iteration algorithm and then simplify various points of operation dramatically. The resulting process is lightweight, requiring little in the way of agent memory or computation and drawing update steps entirely from agent experience. The simplification process points to a number of possible "intermediate" steps between optimistic policy iteration and regret learning proposed. These are discussed briefly but not explored in detail. Selecting between variations of a model of dynamic learning is an empirical question, and although I discuss the method which can address this in the conclusion I leave the actual estimation and model selection for future work. For purposes of exposition this paper instead explores the simplest version of regret learning which contains all elements which are key to capturing both passive, backward-looking behavior, the potential for forward-looking behavior, and which allow agents to eventually engage in social learning of the type described in Chapter 3.

2.3.2 Policy Iteration Framework

The traditional method of finding the optimal policy and value functions for a dynamic stochastic optimization problem such as (C.5) in dynamic programming[16]. There are two

[16]See Bertsekas (2012) for a thorough discussion of this topic.

major practical methods of applying dynamic programming: value iteration, which constructs a sequence of value functions which converge to the optimal value function (from which an optimal policy function can be obtained) by iteratively applying the mapping (2.13) to an arbitrary initial value function, and policy iteration, which iteratively applies a two-step process: from an initial arbitrary policy function, find the associated value function, and from that value function, find the associated policy function. This leap-frog algorithm produces a sequence of policy functions and a sequence of value functions, both of which converge to their optimal counterparts. There are a number of extensions to both value and policy iteration; I will discuss one particular extension to policy iteration which will provide intuition and motivation for an aspect of regret learning below.

Requisite Notation

Before proceeding I define some necessary notation. If the readers is comfortable with the concepts behind dynamic programming and policy iteration in particular, he/she may skip straight to section (2.3.3) below, and reference back here as needed. Appendix A defines the mathematical context of these ideas fully and rigorously and derives detailed multi-step proofs for convergence of policy iteration and optimistic policy iteration. For sake of exposition the following discussion will eschew the full proof-based style of Appendix A while retaining the mathematical rigor; the reader is encouraged to reference Appendix A if any clarification is needed.

Define the transition function f as the function which maps state m_t, choice c_t, and shock y_{t+1} to the next-period state m_{t+1}. Then the right-hand side of the Bellman equation in problem (C.5) prior to applying the optimization step may be defined as:

$$H(m, c, v) = u(c) + \beta \mathbb{E}\left[v(f(m, c, y))\right]. \tag{2.7}$$

To avoid notational overload on the letter "c" when discussing policy iteration, I will replace the "c" function with a more broadly defined "μ" function, as is also done in

Appendix A. After finishing discussion of policy iteration I will switch back to the specific notation of c^{θ} which pertains to my particular problem. The policy (consumption) function μ maps the state m to a consumption choice: $c_t = \mu(m_t)$, and I define $\mathcal{C}(m)$ as the set of acceptable values implied by the law of motion:

$$\mathcal{C}(m) \equiv \left\{ c \;\; s.t. \;\; 0 \leq \;\; c \leq \left(\frac{y' + \bar{q}}{\mathsf{R}} + m \right) \right\}. \tag{2.8}$$

This is a general statement of the acceptable choice set $\mathcal{C}(m)$; note that it includes the next-period income shock y' and the borrowing constraint $\bar{q}$. If I impose the assumed zero borrowing limit and minimum possible income shock, I arrive at the simplified borrowing constraint which I employ in this paper:

$$\mathcal{C}(m) \equiv \{ c \;\; s.t. \;\; 0 \leq c \leq m \}. \tag{2.9}$$

I can now define two mappings which take an arbitrary value function v and map it into a new value function. Given a policy μ[17] and a value function v, define the policy-specific mapping T_{μ} as:

$$(T_{\mu} v)(m) = H(m, \mu(m), v) \;\; \forall m. \tag{2.10}$$

I can also define the optimal mapping T which is independent of a particular policy function and once again maps a value function v into a new value function:

$$(Tv)(m) = \max_{c \in \mathcal{C}(m)} H(m, c, v) \;\; \forall m. \tag{2.11}$$

[17] As noted in the appendix, I assume this policy is consistent with $\mathcal{C}(m)$; that is, $\mu(m) \in \mathcal{C}(m) \; \forall \; m$.

As can be seen, both mappings T and T_μ produce new value functions denoted Tv and $T_\mu v$, respectively. As above I denote the optimal policy and value functions as c^* and v^*, respectively; I can now re-write problem (C.5) as:

$$v^*(m_t) = (Tv^*)(m_t) = \max_{c_t \in \mathcal{C}(m_t)} H(m_t, c_t, v^*) \ \ \forall m. \tag{2.12}$$

This is the familiar result that the optimal value function v^* is the fixed point of the optimal mapping Tv^*, which may be expressed succinctly as

$$v^* = Tv^*, \tag{2.13}$$

where $\forall_m$ is assumed for simplification of notation. This result also follows from the fact that T is a contraction mapping on a complete space; as such a sequence constructed by applying the mapping T to an arbitrary initial value function v_0, k times will produce the sequence $\left\{T^k v_0\right\}$, which converges to the optimal fixed point value function v^*:

$$\lim_{k \to \infty} T^k v_0 = v^*. \tag{2.14}$$

Given the optimal value function v^* I can construct the expression $H(m, \mu m, v*)$ for any policy function μ; then the optimal policy function is simply the function μ^* which solves:

$$H(m, \mu^*(m), v^*) = \max_{c_t \in \mathcal{C}(m)} H(m, c, v^*) \tag{2.15}$$

$$\Rightarrow \mu^*(m) = \underset{c_t \in \mathcal{C}(m)}{argmax}\ H(m, c, v^*) \quad \forall_m.$$

The definition of mu^* in equations (2.15) can be succinctly expressed as

$$T_{\mu^*} v^* = T v^*. \tag{2.16}$$

Note that the relationship in (2.15) can be constructed for *any* arbitrary value function w, not only for the optimal value function v^*. This produces a not-necessarily-optimal policy function μ^w which is associated with the arbitrary function w:

$$H(m, \mu^w(m), w) = \max_{c_t \in \mathcal{C}(m)} H(m, c, w) \tag{2.17}$$

$$\Rightarrow \mu^w(m) = \underset{c_t \in \mathcal{C}(m)}{argmax}\ H(m, c, w) \quad \forall_m.$$

Like T, T_μ is also a contraction mapping on a complete space[18] and like T each has a unique fixed point, denoted v^μ. That is, for a given policy μ there is a unique fixed point value function v^μ which fulfills,

$$v^\mu = T_\mu v^\mu, \tag{2.18}$$

which can be found be repeatedly applying the mapping T_μ to an arbitrary initial value function to produce a convergent sequence $\left\{T_\mu^k v_0\right\}$ such that:

$$\lim_{k \to \infty} T_\mu^k v_0 = v^\mu. \tag{2.19}$$

Again, this follows from the contraction mapping properties of T and T_μ in these contexts. Equations (2.18) and (2.19) provide a way to construct the fixed points associated with the optimal mapping T and policy-specific mapping T_μ.

[18] See Appendix A for the requisite assumptions and proofs for both statements.

2.3.3 Policy Iteration and Optimistic Policy Iteration

Policy iteration is an intuitive process which "ratchets" the value function associated with an arbitrary initial policy function to the optimal value and policy functions. I can define the policy iteration algorithm using equations (2.18) and (2.16) from above. Start with an arbitrary initial policy function μ^0 then iteratively apply the following two steps:

Policy Iteration (PI):

- Step (1) **Policy Evaluation**: Given policy μ^k, find the unique value function v_{μ^k} which is the fixed point of T_{μ^k} as in equation (2.18):

$$v_{\mu^k} = T_{\mu^k} v_{\mu^k}.$$

- Step (2) **Policy Improvement**: Given value function v_{μ^k}, obtain the new policy μ^{k+1} which is optimal with respect to v_{μ^k}. I can state the definition of μ^{k+1} explicitly as the "greedy" solution with respect to $H(x, u, v_{\mu^k})$, as $\mu^{k+1}(x) = \underset{c \in \mathcal{C}(x)}{argmax}\ H(x, c, v_{\mu^k})\ \forall\ x \in X$, and I can express the implicit definition of μ^{k+1} succinctly as:

$$T_{\mu^{k+1}} v_{\mu^k} = T v_{\mu^k}$$

Repeat this process until the distance between $v_{\mu^{k+1}}$ and v_{μ^k} falls within some acceptable tolerance, or in the case of a finite state and action space, $v_{\mu^{k+1}} = v_{\mu^k}$.

The key step is the policy improvement step: defining the next policy function as optimal with respect to the previous value function. In a technical sense, as described in Appendix A, this acts as a sort of "utility ratchet" which guarantees that the next value function iterate will be ever closer to the true value function.

The following section outlines a variation of policy iteration from dynamic programming called "optimistic policy iteration", a version of policy iteration which employs only partial

fixed-point iterations on the policy improvement step. Remarkably, executing the policy evaluation step incompletely *still guarantees convergence.* As noted in Ljungqvist and Sargent (2012)[19], policy iteration and optimistic policy iteration are often much faster than the equivalent value iteration solution for the same problem. As described in Appendix A, the reason for this falls out of the proofs of convergence: the value functions associated with each step in the or both policy iteration and optimistic policy iteration algorithms are bounded *between* those of the value iteration algorithm and the true optimal value function. That is, only in the *worst* case are policy iteration and optimistic policy iteration are as far from the true value function as is value iteration. I turn to that result briefly now.

Optimistic Policy Iteration (OPI):

The optimistic policy iteration algorithm is extremely similar, only slightly modifying the policy evaluation step. The steps occur in the opposite order[20] and notation differs slightly.

Start with an arbitrary initial value function v_0, then iteratively apply the following two steps:

- Step 1 - **Policy Improvement**: Obtain policy μ^k which solves $H(x, \mu^k(x), v_k) = \max_{c \in \mathcal{C}(x)} H(x, c, J_k)$. That is, given v_k, find the policy μ^k consistent with

$$T_{\mu^k} v_k = T v_k.$$

- Step 2 - **Optimistic Policy Evaluation**: Given value function v^k and policy μ_k, execute the optimistic value function update to obtain v_{k+1}, which results from the application of mapping T_{μ^k} to v^k m_k times. That is:

$$v_{k+1} = T_{\mu^k}^{m_k} v_k.$$

[19] Ljungqvist and Sargent (2012) use the term "modified policy iteration." I prefer the Bertsekas (2013) term "optimistic policy iteration" and use it here.

[20] However see Appendix A for a derivation of conditions on v_0 such that either order of steps can be proved to converge

Repeat the process at step 1 using the new value function v_k until convergence.

Note that as with policy iteration above, the policy improvement step can be satisfied by constructing μ^k as the "greedy" policy with respect to $H(x, c, v_k)$:

$$\mu^k(x) = \underset{c \in \mathcal{C}(x)}{argmax}\ H(x, c, v_k)\ \forall\ x \in X.$$

Optimistic policy iteration algorithm differs from policy iteration in the policy evaluation step: instead of iterating to the fixed point of mapping (2.18), only a finite number of mappings, m_k, are applied to the value function at the beginning of each policy evaluation step k. Surprisingly, this process still produces a sequence of value and policy functions which converge to the optimal functions. As with policy iteration, the optimistic policy value function created at each step k is squeezed to the optimal value function by the equivalent steps in value iteration – once again value iteration is a worst-case bound on convergence.

This variation on policy iteration is highlighted for two reasons: first, to show that variations on policy iteration which do not fully estimate v^θ at each step can none-the-less converge to the optimal solution (an intuitive motivation for the value estimation step of regret learning), and second, as an outline for future extensions of regret learning along lines similar to optimistic policy iteration, to be discussed further in the conclusion.

2.4 Regret Learning Solution Method

Regret learning is a form of policy iteration which replaces both the fixed-point calculation in the policy evaluation step, expression (2.3.3), and the policy update step, expression (2.3.3), with approximations constructed from a single agent's experience. In doing so, I impose on the agent the need to *optimize over time.* Rather than optimizing instantaneous each period, the regret learning agent will optimize *eventually*, meandering about in a well-defined neighborhood around the optimal solution, but the agent will not in fact attain this

solution immediately.

2.4.1 Inherent Difficulty of Learning to Optimize from Experience

Before diving into the details of the approximations, I take a moment to think about the difficulties facing an agent attempting to form these estimations from experience. As noted by Allen and Carroll (2001) and Sutton and Barto (1998), the simplest way to approximate the value function associated with a given policy is via Monte Carlo estimation of the sum in expression (2.3), repeated here for convenience:

$$v^{\theta}(m_0) = \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c^{\theta}(m_t)) \right] \tag{2.20}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$$

$$c_t, m_t \geq 0; \;\; m_0 \; given.$$

As luck would have it, agents have access to a data-generating process for the y_t process: their own income experience.[21]

However there are two immediate problems with agents using own experience to estimate v^{θ} as a step in policy iteration. First, agents clearly cannot estimate $v^{\theta}(m)$ for infinite periods – this would preclude them from ever advancing past the first policy evaluation step to the first policy improvement step. Second, agents cannot possibly hope to form an unbiased estimator of the expectation in problem (2.20) in a traditional Monte Carlo sense. To see why, first consider what an ideal method of moments Monte Carlo estimator based on agent experience would look like.

Denote the entire income experience of an agent i as

[21] In the simplest version of regret learning presented here, I restrict agents to only have access to their own income streams – they can learn nothing from other agents who may experience similar shocks.

$$\vec{y} = [y_0, y_1, ..., y_t, ...] \subset \mathbb{R}^{\infty},$$

and denote a k^{th} subset of this income experience, which I will call an "episode," as

$$\vec{y}_k = \left[y_{t_k}, y_{t_k+1}, ..., y_{t_{k+1}-1}\right],$$

where a subset of equidistantly-spaced time indices $tau = \{t_k\}_{k=0}^{\infty},\ k \in \mathbb{N}$ define the borders of subsets of time experience and $D \equiv (t_{k+1} - t_k)\ \forall_k$. Form a "single stream" estimate of v^{θ} in expression (2.20) using one of these k^{th} subsets:

$$w_k^{\theta}(m) = \sum_{t=0}^{D-1} \beta^t u(c^{\theta}(m_t)), \tag{2.21}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$$

$$c_t, m_t \geq 0; \quad m_0 = m\ given.$$

To form a method of moments estimate of the expectation in expression (2.20), ideally the agent would be able to repeat this experience independently many times, for many different starting values of m – say, K times for every $m \in M$. Then the agent i could form the following Monte Carlo estimator for v^{theta}:

$$\bar{w}^{\theta}(m) = \frac{1}{D} \sum_{k=0}^{K} w_k^{\theta}(m_0)\ \ \forall_m \in M. \tag{2.22}$$

Note that there are three dimensions which the agent needs to send to infinity to have the $\bar{w}^{\theta}$ expression be a consistent and unbiased estimator for $v^{\theta}(m)$. It is clear that I need

both $D \to \infty$ and $K \to \infty$ – that is, I need the *length of each episode* to go to infinity, as well as the *number of episodes per state* m to go to infinity as well. In addition, since $m \in \mathbb{R}$, I need the number of starting values of m, N_m, to go to infinity as well.

The problem here is somewhat analogous to the "curse of dimensionality" in traditional dynamic programming, only here it is a "curse of temporality" – the agent who uses own experience as a Monte Carlo data generating process simply *does not have enough time* to form a "good" estimate of the value function v^θ. Finally, *even if* the agent had the time to do this, there is an additional difficulty. To achieve a consistent and unbiased Monte Carlo estimator of $v^\theta(m)$ I need some iid properties. I need the initial m_0 values at which each episodic estimate of $w_k^\theta(m_0)$ is estimated to be iid.[22] I also need the income episodes themselves, $\vec{y}_k$, to be iid in k; while this property is fulfilled for my current problem it will not be fulfilled in general.

Importantly, all of these considerations are only for the first step in the first iteration of policy iteration. If I want to execute policy iteration using agent experience as the data generating process, I must be able to repeat the above steps *indefinitely* until the sequence of policy and value functions converge. The curse of temporality for estimating value functions from an agent's experience appears unavoidably intractable.

There is hope, however; each of the points above is not as insurmountable as they first appear. First, optimistic policy iteration tells us that the exact estimation of v^θ is not needed at each iteration of the policy iteration algorithm – in fact the fixed point calculation in step 1 can be curtailed far short of achieving the estimate of v^θ and convergence is still assured! The key here is that the policy improvement step "ratchets" the next-period policy towards the optimal policy in utility terms, conditional on the current value function, regardless of what that current value function is.

The first-pass regret-learning agents presented here will rely on a similar mechanism. Instead of the partial fixed-point calculation of optimistic policy iteration seen in equation

[22] Ideally the initial m values are distributed as the ergodic distribution of m under the rule θ, presuming that distribution exists

(2.3.3) for OPI, regret learning agents will implement a noisy estimation of the fixed point value calculation in vanilla PI, equation (2.3.3), following a process similar to the one described in equation (2.21).

In order to implement the second policy improvement step, equation (2.3.3), the agent will need to estimate the value function for the entire state space of m. Fortunately the agent will not need to implement estimation (2.3.3) for an infinite number of points to obtain an effective representation of the function $v^{\theta}(m)\ \forall m$; instead the agent will simply estimate a version of this for a grid of points over the state space, and interpolate between these points. As noted in Pál and Stachurski (2013) and Stachurski (2009), it is known that a non-expansive function approximation, such as linear interpolation or nearest neighbor interpolation, will preserve the contraction properties of the T mapping, and thus can preserve convergence properties as well.[23] The agent using regret learning will use a form of linear approximation which is constructed to be convergent in expectation to the function v^{θ}, employing the law of total probability. This form of approximation is motivated by the observation that human decision-makers often divide the state-space of dynamic optimization problems into discrete "chunks" to make the problem tractable, and then proceed with solution (Vanderbilt, 2013). Informally, this "chunking" process can be seen in many economic papers, macroeconomic papers in particular, wherein the dynamic process for, say, the employment matching technology, or house prices, is divided into a discrete set of states, usually an odd number, and usually given labels such as "Good, Average, Bad" (for three states) or "Good, Medium-Good, Average, Medium-Bad, Bad".[24]

The next sections describe the key approximations needed to implement regret learning's experience-based version of policy iteration. The four approximations which will make it possible to solve an approximate version of policy iteration are as follows:

[23] This formal result has been demonstrated in practice for some time. As noted in Carroll (2012b) however, one must be very cautious about errors growing outside of the approximation grid defined by the researcher. The interpolation procedure proposed here in part seeks to address these concerns via an endogenous selection of the grid such that the highest-probability observations as determined from experience occur in exactly the most well-approximated locations.

[24] See for example the aggregate shock process to income in Krusell and Smith Jr (1998).

- determining the state-space partition,
- single-stream estimation of v^{θ},
- identifying regret choices, and
- updating the consumption function by minimizing regret.

These will each be discussed in turn in the following sections, before a complete definition of regret learning is finally provided.

Finally, observe that the *actually experienced* set of states for an episode k can be expressed,

$$\vec{m}_k = \left[m_{t_k}, m_{t_k+1}, ..., m_{t_{k+1}-1}\right],$$

These values are the actual cash-on-hand values produced by the law of motion for episode k under a consumption function c^{θ}; that is:

$$\vec{m}^k_{t+1} = \mathsf{R}(\vec{m}^k_t - c^{\theta}(\vec{m}^k_t)) + y_{t+1},$$

$$y_{t+1} \in \vec{y}_k.$$

This vector of experienced states will be essential to the construction of the state-space partition in the following section.

2.4.2 Determining the State-Space Partition

A key element to regret learning is determining a equiprobable partition of the state space at the end of an episode k of agent experience. The agent will actually use this in two distinct ways.

First, the agent will create a partition of the state space which he/she will use to represent a conditional estimate of the value function for their episode of experience, arriving at an expression like the following, defined for each partition $b_{k,n}$:

$$\hat{v}^{\theta}(m \mid b_{k,n}).$$

The set $b_{k,n}$ is an element of the partition set to be discussed in this section. Importantly, each partition set is constructed to have an equal probability of occurring as seen from agent experience. The condition value function above (defined for each partition) will be constructed in detail in Section (2.4.3). Its primary purpose is to represent the value of falling into each of the possible partitions as seen from experience, which is shaped both by the current shocks as well as the current consumption function. The coarse, equiprobable partition is key to this value function construction.

In the second usage, the agent will use this partition along with:

- the distribution of income experiences in an episode,
- the law of motion of cash-on-hand m_{t+1}, which includes the current consumption choice c_t, and
- the current-period m_t value,

to form a conditional probability of each partition occurring in a *following* period. That is, the probability

$$prob(m_{t+1} \in b_{k,n} \mid c_t, m_t).$$

This is a true conditional distribution based on the current state m_t, as I will see in Section ({sec:forming-the-condition-next-period-partition-prob}). The explicit purpose of forming this probability, the full vector of which will be denoted $\widehat{P}_m^k$, is to be able to form a real-valued empirical estimate of the conditional value function arising at state m_t under choice c_t. I am forming an experienced-based version of the H function defined in expression (2.7) above. The H function is key for policy iteration, and its empirical counterpart will be key for regret learning.

I want to end up with an expected conditional value function which is dependent on the

current state and agent choices:

$$\hat{v}^{\theta}(c_t, m_t) = \hat{v}^{\theta}(m_{t+1} \mid b_k) \bullet \widehat{P}_m^k = \sum_{n=1}^{N} \left(\hat{v}^{\theta}(m_{t+1} \mid b_{k,n}) prob(m_{t+1} \in b_{k,n} \mid c_t, m_t) \right).$$

Those familiar with Q-learning will recognize the left-hand side of the this expression as appearing surprisingly similar to the traditional two-state "Q factors" used in Q-learning. Q-learning is an online learning-top-optimize solution to dynamic programming problems in reinforcement learning and approximate dynamic programming, with a long history and a successful application to many practical real-world problems (see Powell (2007) and Sutton and Barto (1998) and references therein). A difficulty with Q-learning is that it is most easily employed over a finite state and action space – something encountered in only a subset of economic problems. Properly interpolating over the space of Q-factors to bring Q-learning into continuous space problems is difficult, and in addition, the choice of stepsize[25] for the Q-factor update is not straightforward. Regret learning sidesteps both the finite space issue and the stepsize issue entirely via the partitioning to be discussed in this section.

This expression (2.4.2) uses the law of total probability to form the final conditional value function estimate. As can be seen, this function varies by changing the *conditional probabilities* of arriving at each partition tomorrow, which itself varies with the choice of c_t. Thus I use a carefully constructed *fixed* estimate of the value function over a set of partitions with a *variable* conditional distribution function to produce to full variable conditional value function. The law of total probability motivates this step, and the key important justification for employing the law of total probability here is the usage of the fixed partition over the state space. As will become evident, even slight amounts of agent experience, excluding any externally acquired information, can emulate the entire policy iteration process effectively, providing quick convergence to a well-defined function space around the optimal

[25]Roughly equivalent to a "gain" parameter in the more traditional macroeconomic learning literature, see Evans and Honkapohja (2001) for more detail.

solution (in utility terms), while still maintaining an intuitively appealing process of the agent being "pushed around" the function space by streams of good or bad experience.

All of this, however, is yet to be discussed in the results, Section (2.5). I must first work through the construction of the state-space partition and a few other concepts.

The coarse partitioning is motivated by the idea that human behavior, in attempting to solve complex optimization problems, appears to involve "chunking" of the problem space into self-similar partitions to offer more efficient solution of these approximate problems (Vanderbilt, 2013). This approach is also widely taken in approximate dynamic programming and reinforcement learning in computer science; see Powell (2007) and Sutton and Barto (1998) for discussion and references. The partitioning I use here is a simple implementation of this idea for a consumption-savings problem.

The second related motivation behind partitioning the state space is the idea that the agents need to capture the shape of their value function, but by necessity they have few observations to use for this estimation. Or rather, the cost of obtaining additional observations is very high. There is a need for efficient use of the few points of experience the agent obtains because there is a tradeoff between a very fine partition, which requires extensive amounts of experience simply to populate much less achieve a good estimate, and a very course partition, which is populated quickly but may form a more poor approximation to the true function.

In addition, I want the partition system, and the points chosen to represent each partition, to endogenously reflect the experience the agent has had – for better or for worse. If the agent encounters an unusual set of income shocks, I want the partition of the state space to endogenously reflect this. This is achieved via the simple equiprobable partitioning of the state space based on the agent's experience. As with all steps, this partition is construct from experience in such a way that it will converge to an appropriate object as learning time goes to infinity.

Denote the cumulative density function of the state variables m under a particular consumption function c^θ (and under a particular income process) as $\mathcal{F}_m^\theta$, and denote its

inverse, the quantile function, as $\mathcal{Q}_m^\theta \equiv \mathcal{F}_m^{-1,\,\theta}$. Denote the empirical counterparts to both of these, constructed with sample size D, as $\hat{\mathcal{F}}_{m,D}^\theta$, and $\hat{\mathcal{Q}}_{m,D}^\theta$, respectively. Both empirical counterparts converge to the true functions as $D \to \infty$.[26]

I can now use the D-length vector of cash-on-hand states experienced in episode k, $\vec{m}_k$, to find an equiprobable partition of the state space. Given N, the number of partitions, simply find the partition boundaries for episode k as follows:

$$B_k \equiv [B_{k,0}, B_{k,1}, ..., B_{k,n}, ..., B_{k,N}]\,, \text{with elements defined,}$$

$$B_{k,n} = \hat{\mathcal{Q}}_{m,D}^\theta \left(\frac{n}{N}\right),\ for\ n = 0, 1, ..., N.$$

In addition, denote the partitions defined by the boundaries B_{k} as follows:

$$b_{k,n} = [B_{k,n-1}, B_{k,n})\ for\ n = 1, 2, ..., N.$$

The number of boundary elements in B_k is $N+1$, and the number of partitions bounded by B_k is N. For convenience, denote the finite sets which are defined by $b_{k,n}$ and $\vec{m}_k$ as

$$A_k \equiv [A_{k,1}, ..., A_{k,n}, ..., A_{k,N}]\,,\ \text{where,}$$

$$A_{k,n} \equiv \{m\ s.t.\ m \in \vec{m}_k, m \in b_{k,n}]\,.$$

Because of the discrete nature of the empirical density used, the probability that each partition occurs is not exactly $\frac{1}{N}$ but rather,

$$p_k = [p_{k,1}, ..., p_{k,n}, ...p_{k,N}]\,,\ \ where$$

$$p_{k,n} = \frac{\#A_{k,n}}{D}.$$

[26] While the empirical cumulative density function (ECDF) has a unique form, there are a number of way to construct its inverse, the empirical quantile function, due to the step-function nature of the ECDF. I employ the median-unbiased quantile function estimator recommended by Hyndman and Fan (1996).

The operator # denotes the number of elements in set $A_{k,n}$. Now the key question is how to choose the point to represent each partition. The straightforward answer is to define the grid $\hat{M}^k$ as the empirical conditional expected value of each partition $A_{k,n}$:

$$\hat{M}^k = [M_{k,1}, ..., M_{k,n}, ..., M_{k,N}], \text{ where,}$$

$$M_{k,n} = \frac{1}{\#A_{k,n}} \sum_{m \in A_{k,n}} m.$$

where each $M_{k,n}$ converges to the conditional expectation $\mathbb{E}[m \mid m \in A_{k,n}]$ as $D \to \infty$.

Figure (2.10) provides a visualization of this process for a hypothetical agent who has experienced 25 states in an episode ($D = 25$) and who has 5 coarse partitions to summarize experience ($N = 5$). The solid red vertical lines represent the boundaries of the partitions, B_k; the dashed blue lines indicate the conditional means of each bin, $\hat{M}^k$. The dotted red lines demonstrate that the space has indeed been partitioned into sections with equal probability of occur according to the empirical quantile function $\hat{\mathcal{Q}}^{\theta}_{m,D}$.

Note that, by construction,

$$\frac{1}{N} \sum_{m \in \vec{m}_k} m = M_{k,n} \bullet p_{k,n},$$

which is the empirical counterpart to one simple application of the law of total probability.

2.4.3 Single-Stream Estimation of v^{θ_k}

Consider again a single "episode" k of income experience for agent i, with episode length D as described above:

$$\vec{y}_k = \left[y_{t_k}, y_{t_k+1}, ..., y_{t_{k+1}-1}\right].$$

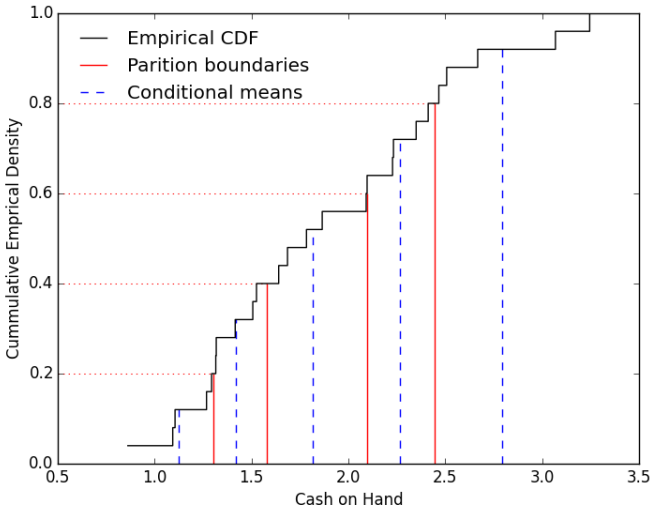


Figure 2.10: Partitioning the m Space: 5 bins, $D = 25$ learning periods

Assume that the agent would like to execute the estimator of v^{θ_k} presented in expression (2.21) using this income experience $\vec{y}_k$ for a grid of points in the state space. The agent does not calculate this estimate for just any set of points in the state space, but rather very specifically on the grid defined by $\hat{M}^k$.

The agent is in fact attempting to calculate the empirical equivalent to the following:

$$v^{\theta}(m \mid b_{k,n}) \equiv [E]\left[v^{\theta}(m) \mid m \in b_{k,n}\right],$$

where I will denote the empirical equivalent:

$$\hat{v}^{\theta}(m \mid b_{k,n}) \equiv [\hat{E}]\left[v^{\theta}(m) \mid m \in b_{k,n}\right],$$

abusing notation slightly by using "$v(m \mid b_{k,n})$" as shorthand for "$v(m \mid m \in b_{k,n})$."

I estimate this conditional expectation version of the value function to take advantage of the law of total probability later, to allow agents to calculate a dynamic estimate of

next-period value when learning about regret choices.

Executing the single-stream estimate for each point in the grid is now deceptively straightforward. The agent simply maintains a vector of state variables, $\vec{m}^k_{N,t}$, which is of length N and initially set to the grid for episode k:

$$\vec{m}^k_{N,0} \equiv \hat{M}^k.$$

The N subscript acts to remind us that this particular vector has dimensions $N \times 1$, that is, it is the length of the grid on which I am estimating the value function, instead of $D \times 1$, which is the time-series length of the current episode k.

This vector of m-values is then updated following the law of motion implied by the current policy function c^θ and the shocks $\vec{y}_k$

$$\vec{m}^k_{N,t+1} = \mathsf{R}(\vec{m}^k_{N,t} - c^\theta(\vec{m}^k_{N,t})) + y_{t+1},$$

$$y_{t+1} \in \vec{y}_k.$$

The agent also maintains a rolling summation of discounted utility experienced following policy $c^{theta}(.)$. Let $\vec{w}^{k,\theta}_t$ denote the vector of value function estimates, updated similarly each period:

$$\vec{w}^{k,\theta}_{t+1} = \vec{w}^{k,\theta}_t + beta^t u(c^\theta(\vec{m}^k_{N,t}))$$

$$\vec{w}^{k,\theta}_0 = u(c^\theta(\vec{m}^k_0)),$$

which produces the final vector

$$\vec{w}^\theta_k \equiv \vec{w}_{t_{k+1}-1}.$$

This final vector $\vec{w}_k^\theta$ has been constructed to represent $\hat{v}^\theta(m \mid b_{k,n})$. Importantly, because of the nature of its construction in equations (2.4.3), the monotonicity of the original utility function is preserved as the estimation progresses. As discussed in the appendix, monotonicity plays a key role in the convergence properties of policy iteration, and I strive to maintain the monotonicity of elements of the regret-learning value function whenever possible.

Note that this conditional value function vector can be dotted with the empirical probabilities of each partition occurring, to arrive at an unconditional point estimate of the expected value of following the consumption function $\hat{c}^\theta$. This will be important for combining this individual-level learning of this paper with the social learning suggested in Chapter 3. This important advancement will be discussed further in the conclusion.

2.4.4 Identifying Regret Choices

Once an agent has experienced D periods in an episode k, determined the state-space partition $\hat{M}^k$ and formed a conditional value function estimate $\hat{v}^\theta(m \mid b_{k,n})$ for that episode, it's time to look back and think about what consumption choices the agent *should have made* conditional on $\hat{v}^\theta(m \mid b_{k,n})$.

Forming Conditional Distribution $prob(m_{t+1} \in b_{k,n} \mid c_t, m_t)$

I must first outline a key element: forming the one-period-ahead distribution of the occurrence of each partition. Recall the partition boundaries for this episode k:

$$B_k \equiv [B_{k,0}, B_{k,1}, ..., B_{k,n}, ..., B_{k,N}], \text{with elements defined,}$$

$$B_{k,n} = \hat{\mathcal{Q}}^\theta_{m,D}(\frac{n}{N}),$$

and recall that these boundaries denote the partitions:

$$b_{k,n} = [B_{k,n-1}, B_{k,n}) \, for \; n = 1, 2, ..., N.$$

As with the state variables m, denote the cumulative density function of the income shock as $\mathcal{F}_y$, and denote its empirical counterpart, constructed with sample size D, as $\hat{\mathcal{F}}_{y,D}$. For an episode k I will construct an empirical cumulative (ECDF)for income using $\vec{y}_k$; denote this specific empirical density function $\hat{\mathcal{F}}^k_{y,D}$.

For each partition $b_{k,n}$, denote the probability that *next-period* cash-on-hand m_{t+1} will fall in that partition, conditional on current-period cash-on-hand m_t, as:

$$q(b_{k,n} \mid c_t, m_t) \equiv prob(m_{t+1} \in b_{k,n} \mid m_t).$$

I can define this as follows. Consider the problem for a single partition

$$b_{k,n} = [B_{k,n-1}, B_{k,n}) \, ,$$

and recall the law of motion for m under the consumption choice c_t is

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}.$$

Then,

$$\begin{aligned} prob(m_t + 1 \in b_{k,n} \mid m_t) &\equiv prob(B_{k,n-1} \leq m_{t+1} < B_{k,n}) \\ &= \mathcal{F}^{\theta}_m(B_{k,n} \mid m_t) - \mathcal{F}^{\theta}_m(B_{k,n-1} B_{k,n} \mid m_t). \end{aligned}$$

Note that:

$$\begin{aligned}\mathcal{F}_m^\theta(B_{k,n} \mid m_t) &= prob(\mathsf{R}(m_t - c_t) + y_{t+1} \leq B_{k,n}) \\ &= prob(y_{t+1} \leq B_{k,n} - \mathsf{R}(m_t - c_t)) \\ &= \mathcal{F}_y(B_{k,n} - \mathsf{R}(m_t - c_t)),\end{aligned}$$

Define $z_n(c_t \mid m_t) \equiv B_{k,n} - \mathsf{R}(m_t - c_t)$. Now,

$$\begin{aligned}q(b_{k,n} \mid c_t, m_t) &\equiv prob(m_{t+1} \in b_{k,n} \mid c_t, m_t) \\ &= prob(B_{k,n-1} \leq m_{t+1} < B_{k,n}) \\ &= \mathcal{F}_m^\theta(B_{k,n} \mid m_t) - \mathcal{F}_m^\theta(B_{k,n-1}B_{k,n} \mid m_t) \\ &= \mathcal{F}_y(z_n(c_t \mid m_t)) - \mathcal{F}_y(z_{n-1}(c_t \mid m_t)) \\ &\approx \hat{\mathcal{F}}_{y,D}(z_n(c_t \mid m_t)) - \hat{\mathcal{F}}_{y,D}(z_{n-1}(c_t \mid m_t)).\end{aligned}$$

Thus I can construct the probability that m_{t+1} falls into the partition $b_{k,n}$ as

$$q(b_{k,n} \mid c_t, m_t) = \mathcal{F}_y(z_n(c_t \mid m_t)) - \mathcal{F}_y(z_{n-1}(c_t \mid m_t)),$$

and the empirical equivalent estimated from experience as,

$$\hat{q}(m_{t+1} \in b_{k,n} \mid c_t, m_t) = \hat{\mathcal{F}}_{y,D}(z_n(c_t \mid m_t)) - \hat{\mathcal{F}}_{y,D}(z_{n-1}(c_t \mid m_t)),$$

a straightforward calculation. I will abbreviate this expression

$$\hat{q}(b_{k,n} \mid c_t, m_t)$$

for notational convenience. Denote the full partitioned probability mass function for

episode k, which uses $\hat{\mathcal{F}}^k_{y,D}$, as

$$\widehat{P}^k_m(c_t \mid m_t) = [\hat{q}(b_{k,1} \mid c_t, m_t), ..., \hat{q}(b_{k,n} \mid c_t, m_t), ..., \hat{q}(b_{k,N} \mid c_t, m_t)] \,.$$

The notation "$(c_t \mid m_t)$" is intended to communicate the idea that m_t will given by nature, while c_t is chosen. This distinction will become apparent in the next section.

Next, construct the empirical, learned-from-experience version of expression (2.7) above:

$$\begin{aligned}
\hat{H}(m, c, \hat{v}^\theta) &= u(c) + \beta(\hat{v}^\theta(m \mid b_{k,n}) \bullet \widehat{P}^k_m(c \mid m)) \\
&= u(c) + \beta \sum_{n=1}^{N} \left(\hat{v}^\theta(m \mid b_{k,n}) \times \hat{q}(b_{k,n} \right) \\
&= u(c) + \beta \sum_{n=1}^{N} \left([\hat{E}] \left[v^\theta(m) \mid m \in b_{k,n} \right] \times \hat{q}(b_{k,n} \right) .
\end{aligned}$$

The goal of this expression is to come as close as possible to constructing an empirical expression for $\mathbb{E}\left\{v^\theta(m_{t+1}) \mid c_t, m_t\right\}$ using an empirical equivalent to the law of total probability. This provides a rigorous framework for expressing the idea that the agent learns imperfectly about both the value of states as well as the dynamics of their environment from experience.

I am now ready to formalize "regret" and "learning from regret."

Identifying Regret Choices

The agent can now identify "what they should have done" conditional on the learned value function $\hat{v}^\theta(m \mid b_{k,n})$. Assume that the agent has been following consumption function c^θ for episode k. Recall that the agent's experience in episode k is formalized in the following two vectors,

$$\vec{y}_k = \left[y_{t_k}, y_{t_k+1}, ..., y_{t_{k+1}-1}\right], \text{ income shocks, and}$$

$$\vec{m}_k = \left[m_{t_k}, m_{t_k+1}, ..., m_{t_{k+1}-1}\right], \text{ cash-on-hand states from following } c^\theta.$$

For each period t in episode k, the agent determines the regret choice $\check{c}_t$:

$$\check{c}_t = \underset{c \in [0, m_t]}{argmax}\ \hat{H}(m_t, c, \hat{v}^\theta),$$

where the full vector of these choices for episode k is defined,

$$\vec{\check{c}}_k \equiv \left[\check{c}_{t_k}, \check{c}_{t_k+1}, ..., \check{c}_{t_{k+1}-1}\right].$$

Note that each of the $\check{c}_{t_k}$ values directly corresponds to a m_{t_k} value. Now with the regret choices in hand, the agent simply needs to find the consumption function which is closest to these values in a mean squared errors sense.

2.4.5 Improving c^θ by Minimizing Regret

Once the regret choices $\vec{\check{c}}_k$ are in hand for episode k, the agent needs to determine the policy function to use in the next period, $k+1$. Recall the structure of the consumption function the agent is using, parameterized by $\theta = (\kappa, \bar{m})$:

$$\widetilde{c}^\theta(m_t) = E[y_t] + \kappa\left[m_t - \bar{m}\right], \text{ and}$$

$$c^\theta(m_t) = \begin{cases} m_t & if\ \widetilde{c}^\theta(m_t) \geq m_t \\ \widetilde{c}^\theta(m_t) & otherwise. \end{cases}$$

where as before κ is the marginal propensity to consume out of wealth, $\bar{m}$ is the buffer-stock savings target, and the piecewise linear form of c^θ imposes the liquidity constraint.

The problem facing the consumer is to find the parameters $\hat{\theta} = (\hat{\kappa}, \hat{\bar{m}})$ such that the sum of squared errors between the parametric portion of the consumption function applied to the experienced states, $\widetilde{c}^{\hat{\theta}}(\vec{m}_k)$, and the regret choices at each of those states, $\vec{\check{c}}_k$, is minimized.[27] Mathematically, the agent wants to solve:

$$\hat{\theta} = \underset{\theta}{argmin} \sum_{t=t_k}^{t_{k+1}-1} \left(\widetilde{c}^{\hat{\theta}}(m_t) - \check{c}_t \right)^2,$$

where each $m_t \in \vec{m}_k$ and $\check{c}_t \in \vec{\check{c}}_k$ as defined in the above section. Implementing this error minimization for the $\widetilde{c}^{\theta}$ portion of the consumption function via OLS is straightforward; run the OLS estimate on:

$$\vec{\check{c}}_k = \alpha_0 + \alpha_1 \vec{m}_k$$

to obtain estimates of the constant α_0 and slope α_1, respectively, and simply back out the $(\hat{\kappa}_k, \hat{\bar{m}}_k)$ values as

$$\hat{\kappa}_k = \alpha_1, \text{ and}$$

$$\hat{\bar{m}}_k = \frac{E[y] - \alpha_0}{\hat{\kappa}_k},$$

which can be seen by simply rearranging the definition of $\widetilde{c}^{\theta}$:

[27] The decision whether to use $\widetilde{c}^{\theta}$ versus c^{θ} to minimize distance to the regret choices is simply due to computational convenience and reduction in programming error. Future versions will include a restricted regression form which imposes the liquidity constraint directly. Initial experiments with versions of both forms indicates that the outcomes of either approach are extremely similar.

$$\widetilde{c}^{\theta}(m) = E[y] + \kappa\left[m - \bar{m}\right],$$

$$= (E[y] - \kappa\bar{m}) + (\kappa m)$$

$$= \alpha_0 + \alpha_1.$$

Note that technically the value $E[y_t]$ is a parameter of the consumption function in equation (2.4.5), unless the agent knows this value perfectly. Since the agent must learn everything from experience, the agent learns this value as well. Instead of estimating $E[y_t]$ in the same way that $\hat{\theta}$ is estimated, the agent simply "calibrates" $E[y_t]$:

$$E[y] = \frac{1}{D}\sum_{y \in \vec{y}_k} y.$$

The resulting consumption function can be denoted $c^{\theta_{k+1}}$, and is set as the consumption function for the following episode:

$$\widetilde{c}^{\theta_{k+1}}(m_t) = E[y_t] + \kappa_k\left[m_t - \bar{m}_k\right], \text{ and}$$

$$c^{\theta_{k+1}}(m_t) = \begin{cases} m_t & if\ \widetilde{c}^{\theta_{k+1}}(m_t) \geq m_t \\ \widetilde{c}^{\theta_{k+1}}(m_t) & otherwise. \end{cases}$$

Figure (2.11) displays a single step of this process for an agent who is learning from regret. The dashed blue line represents the agent's previous-episode consumption function c^{θ_k}. The blue stars represent the regret choices identified above, and the solid blue line represents the new regret-minimizing consumption function $c^{\hat{\theta}}$. The red dashed line represents the true optimal consumption function.

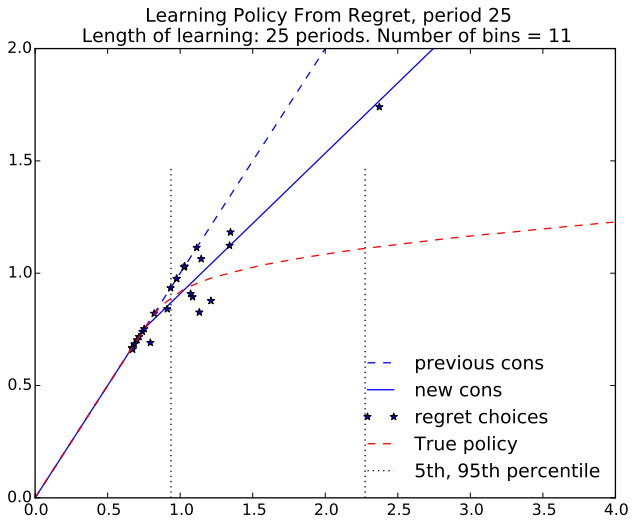


Figure 2.11: Improving the Consumption Function by Minimizing Regret. Agent using 11 bins, $D = 25$ learning periods

One final important note: agents are restricted from learning consumption functions with either an unattainable $\bar{m}$ buffer-stock savings targets (namely, a buffer-stock target outside the borrowing constraint) or a negative slope on the MPC, κ. Without this restriction, an agent may occasionally learn a rule with either property, with the consequence that the agents takes a large step away from the optimal rule, essentially "starting over" the learning process in utility term. However the agent quickly returns to learning appropriate rules in the following episodes; however this brief departure obscures "regular" agent behavior when examining the entire distribution (this will be discussed further below). Both restrictions on $\bar{m}$ and κ seem to be common-sense, and in addition both are supported by theoretical restrictions on these parameters.

2.4.6 Regret Learning

I can now finally concisely outline the regret learning algorithm. Recall the two steps of policy iteration, policy evaluation (2.3.3) and policy improvement (2.3.3), respectively, repeated here for convenience:

$$v_{\mu^k} = T_{\mu^k} v_{\mu^k}, \text{ policy evaluation, and,}$$

$$T_{\mu^{k+1}} v_{\mu^k} = T v_{\mu^k} \text{ policy improvement.}$$

Regret learning also has a policy evaluation step and a policy improvement step. These rely on the approximations discussed in the previous sections, which empirically approximate various components of both steps above using experience. They are as follows.

Start with an arbitrary initial consumption function c_0^θ, then iteratively apply the following two steps for D-length episodes of experience, each episode denoted k:

Regret Learning:

- Step (1) **Policy Evaluation**: Given policy c^{θ_k}, estimate v^{θ_k} from experience as described in Section (2.4.3), as a noisy, single-stream, curvature retaining Monte Carlo

estimate of v^{θ}.

- Step (2) **Policy Improvement**: Given the conditional value function v^{θ_k}, find the regret choices $\vec{c}_k$ as described in Section (2.4.4). Using these regret choices, find the new policy function $c_{k+1}^{\theta'}$ which minimizes the regret choices in a mean-square errors sense, as described in Section (2.4.5).

These steps are repeated indefinitely.

2.4.7 Need for an Agent-Based Simulation

Naturally, there are a number of important questions to ask. Although the approximating components of regret learning have been largely written such that they individually converge to their theoretical counterparts, full convergence to the theoretical optimal policy and value functions has not been proved, nor will it be proved in this paper.[28] I can, however, say quite a bit about the behavior of regret learning through an agent-based simulation. I use this method to sketch out the qualities of regret learning in the following section.

2.5 Results

While the analytical properties of regret learning have yet to be thoroughly explored, I can say quite a bit with numerical simulation. The broad questions I would like to answer are the following:

- Can regret learning attain a near-optimal policy?
- How long does this take, and what does it look like?
- How is behavior affected by the parameters D and N?

To address these questions, 1,000 agents were simulated for 10,000 periods for a variety of combinations of the N, D parameters: the number of partitions used to model the

[28] Completing the analytical foundations for regret learning is a continuing task to be completed in future work.

following period, and the number of periods used to estimate a value function before updating, respectively. The agent's *behavioral* parameters, risk aversion (ρ), discounting (β), and variance of shocks to income (σ_y), are calibrated to common values estimated from microeconomic data. The specific parameters used are expressed in Table (2.1).

The results of the simulation experiment is largely positive. Agents start with a spendthrift consumption function which has a sacrifice value of roughly 0.9, which can be interpreted as equal to a one-time payment of ˜90% of expected annual income. Within one learning episode, this sacrifice value can be cut in half, and within three to five episodes it can be cut to less than a tenth of the original sacrifice value. These improvements are only partially permanent, however. As I will demonstrate below, agents spend a short time improving permanently over their initial, worst-case spendthrift consumption functions, but then spend the rest of their lives buffeted around a well-defined distribution of distances from the optimal rule. The reasons for agent indecision about their learning rules is intuitive – they forget their distant past as they continue to live, and their most recent set of experiences can deceive them about the effectiveness of a consumption rule. As the researchers I a aware of this deception, but to the agent, acting on their conditional and limited information, it appears to be the best choice. I argue that this is a feature, not a bug – while agents can be forced very close to the optimal solution, it is intuitively appealing to have a rigorous model of agent learning, which strives for optimal behavior (and in fact explicitly has optimal behavior as its target) but none-the-less falls prey to whatever sets of shocks the agent has most recently experienced. After examining the characteristics of this behavior I will discuss the particular structural reasons which drive it, and outline extensions which can extend these results.

2.5.1 Aggregate Regret-Learning Behavior

I will examine agent behavior first from a bird-eye view, outlining distributional effects before diving into specific examples to explore the details of the mechanics. I will primarily focus on the differences in behavior which derive from different N and D values. For a given

(N, D) parameter pair, agents converge to a distribution of distances from the optimal rule fairly quickly, within 5-10 episodes (recall that each episode is of length D). This is displayed in Figures (2.12) through (2.22), which show the time series of sacrifice values from start of a simulation through a total of 10,000 periods.

The figures are organized by N-value, the number of partitions the agents use to create the value function estimate. For each N value, the values for a number of lengths of learning period, denoted D, are displayed.

For $N \in [3, 5, 7, 11]$, there is a figure displaying the distribution of sacrifice values $\bar{\epsilon}^{\theta}$ over time for $D \in [13, 24, 48, 72]$ and another figure displaying the distribution of $\bar{\epsilon}^{\theta}$ over time for $D \in [101, 201, 301]$. For $N \in [25, 55, 95]$, there are only figures for $D \in [101, 201, 301]$.[29]

The choice of $N \in [3, 5, 7, 11]$ is to demonstrate the high gains agents experience for increasing N even a little. The choice of $N \in [25, 55, 95]$ demonstrates that these gains quickly start to tail off as N increases, if D is being held constant.

The choices for $D \in [13, 24, 48, 72]$ correspond to roughly the decade after late teens, the two decades after late teens, four decades after late teens, and lastly an entire lifetime after late teens.

Finally, the choices for $D \in [101, 201, 301]$ are again to demonstrate how the marginal gains from an increase in learning lengths drop off as D increases for a fixed N.

The parameters used for plots (2.12) through (2.22) are summarized in Table 2.2.

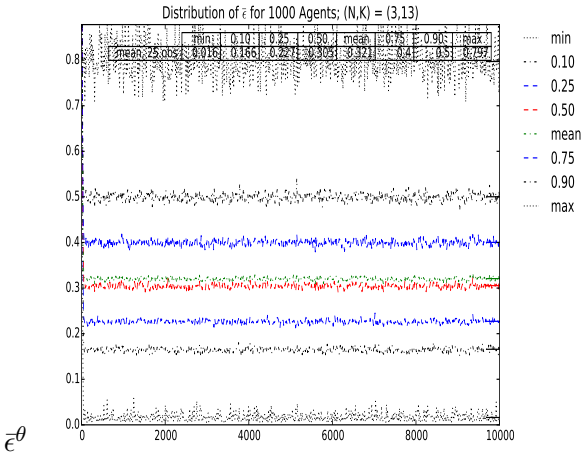
[29]Note that $N < D$ in all circumstances; if this is not the case, agents cannot populate their probability mass function in expression (2.4.4).

Table 2.2: Parameter Sweep Values

Plots in Figures (2.12) through (2.22)

	Number of partitions, N						
Length of learning episode D	3	5	7	11	25	55	95
13 periods	x	x	x	x			
24	x	x	x	x			
48	x	x	x	x			
72	x	x	x	x			
101	x	x	x	x	x	x	x
201	x	x	x	x	x	x	x
301	x	x	x	x	x	x	x

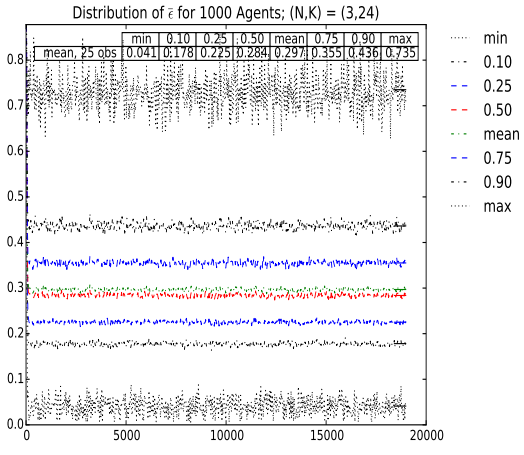
Distribution of $\bar{\epsilon}^{\theta}$ for N = 3, D = 13



$\bar{\epsilon}^{\theta}$

Time

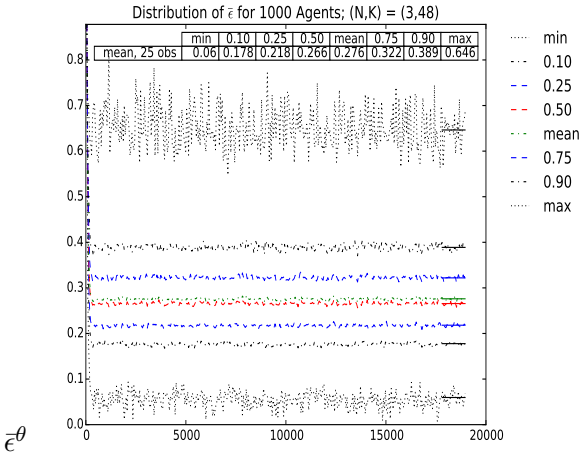
Distribution of $\bar{\epsilon}^{\theta}$ for N = 3, D = 24



$\bar{\epsilon}^{\theta}$

Time

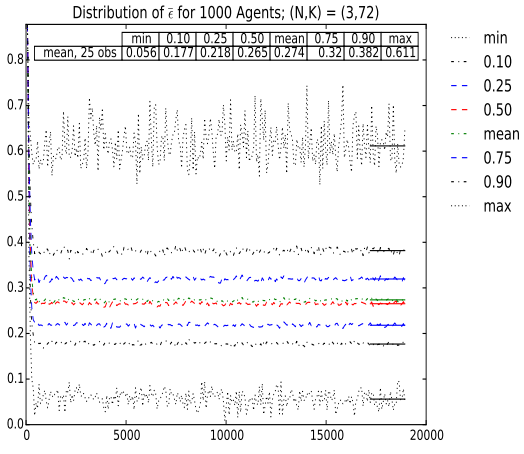
Distribution of $\bar{\epsilon}^{\theta}$ for N = 3, D = 48



$\bar{\epsilon}^{\theta}$

Time

Distribution of $\bar{\epsilon}^{\theta}$ for N = 3, D = 72



$\bar{\epsilon}^{\theta}$

Time

Figure 2.12: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 3, D = 13, 24, 48, 72

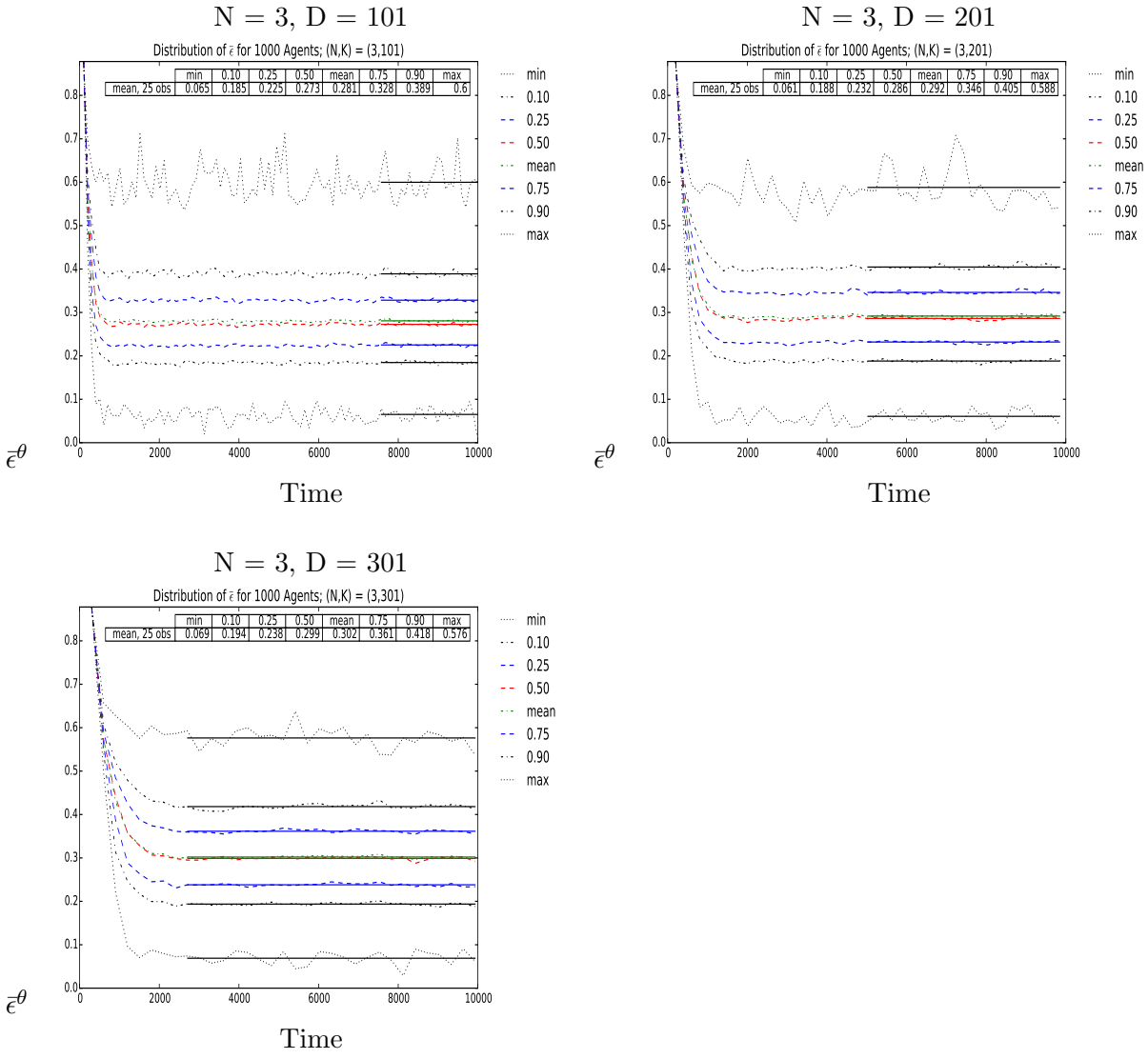


Figure 2.13: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 3, D = 101, 201, 301

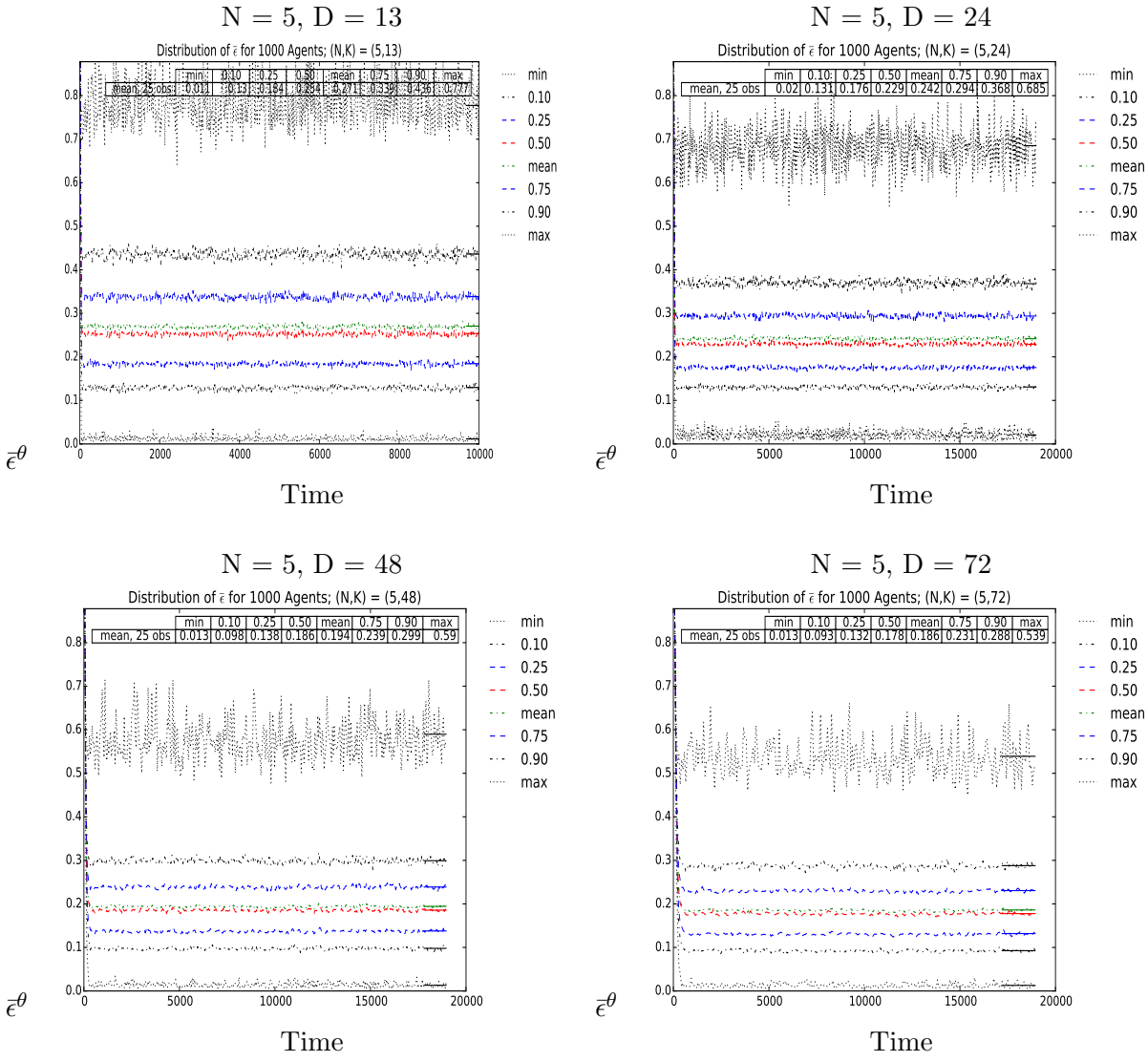


Figure 2.14: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 5, D = 13, 24, 48, 72

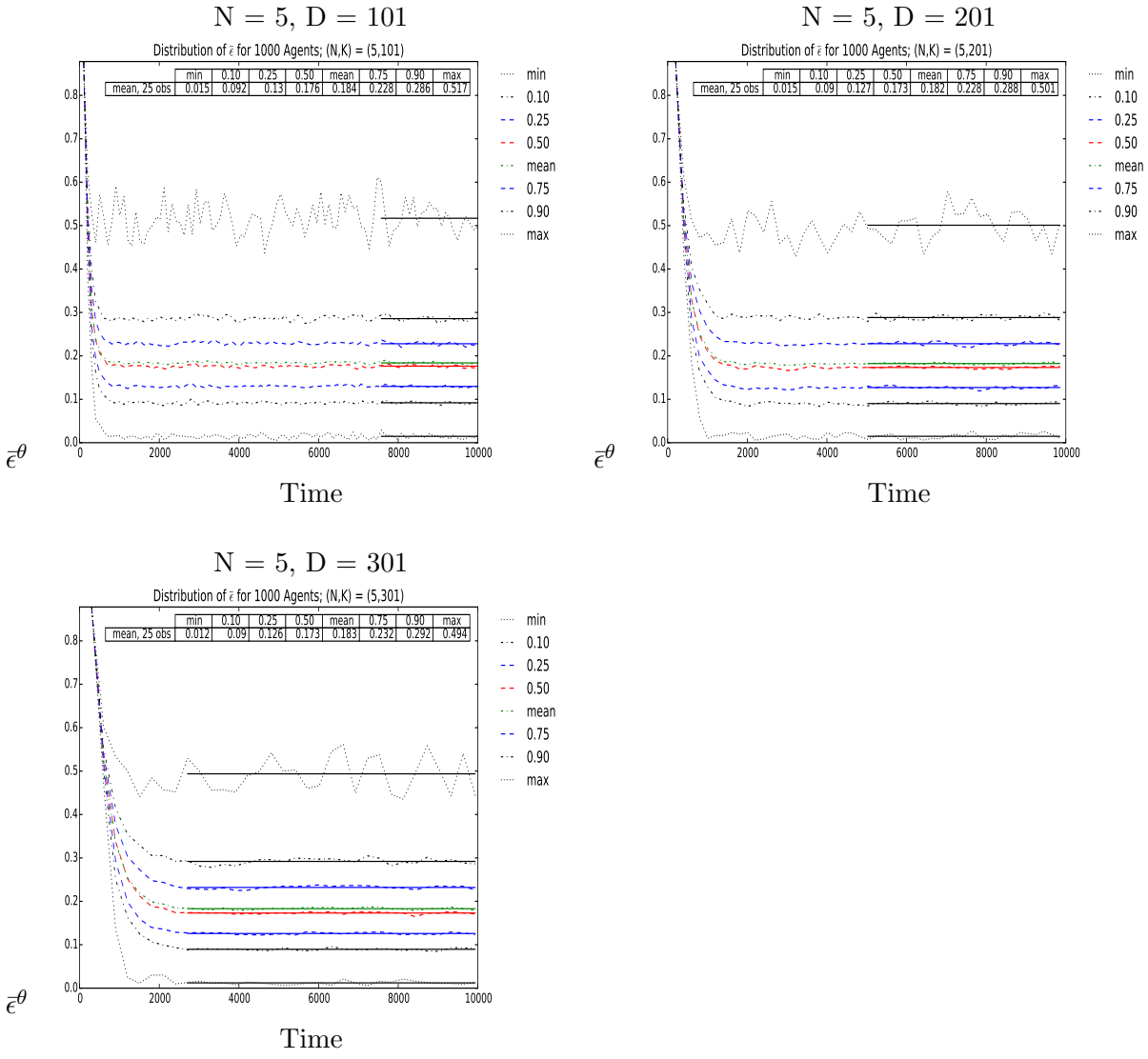


Figure 2.15: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 5, D = 101, 201, 301

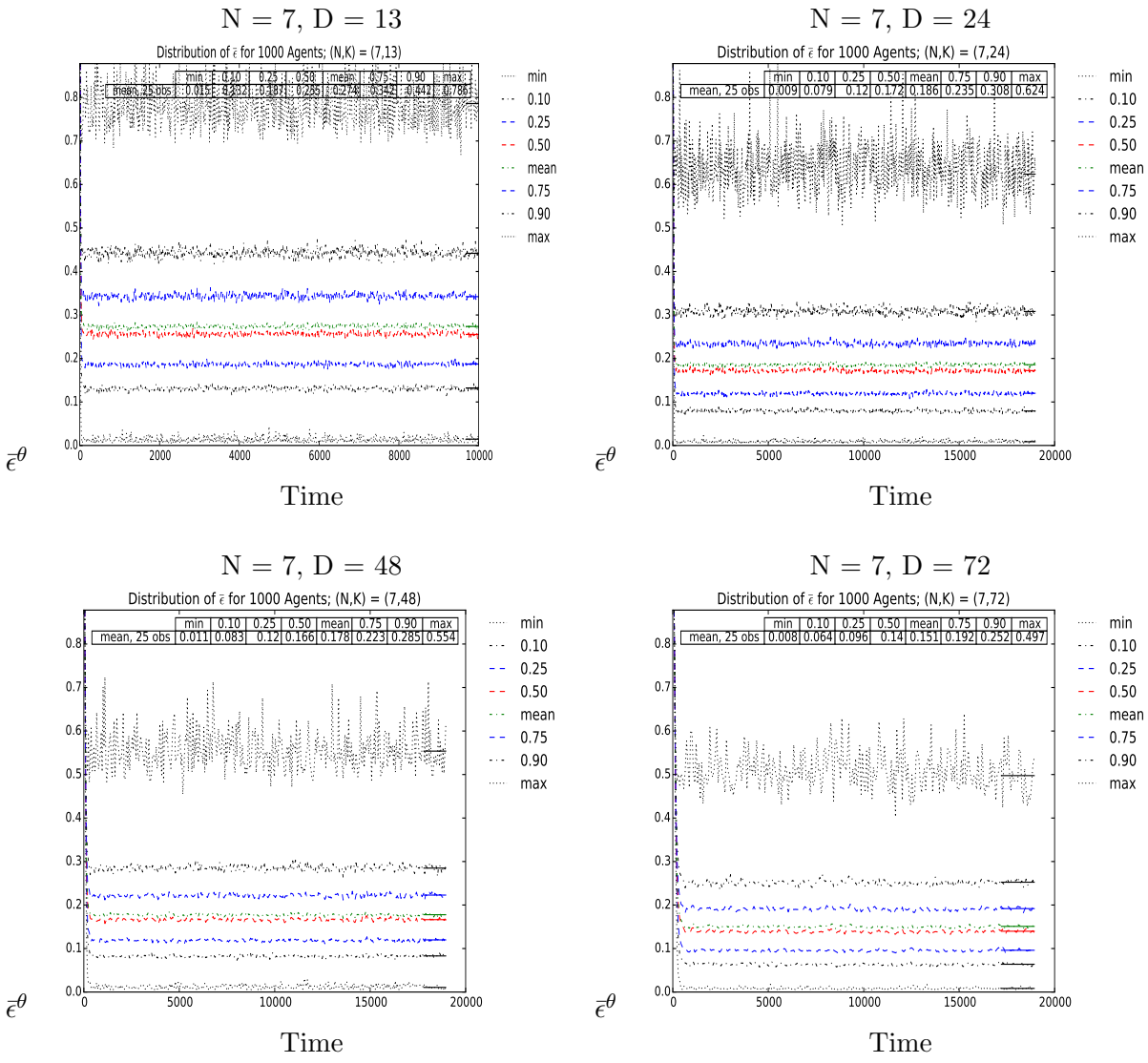


Figure 2.16: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 7, D = 13, 24, 48, 72

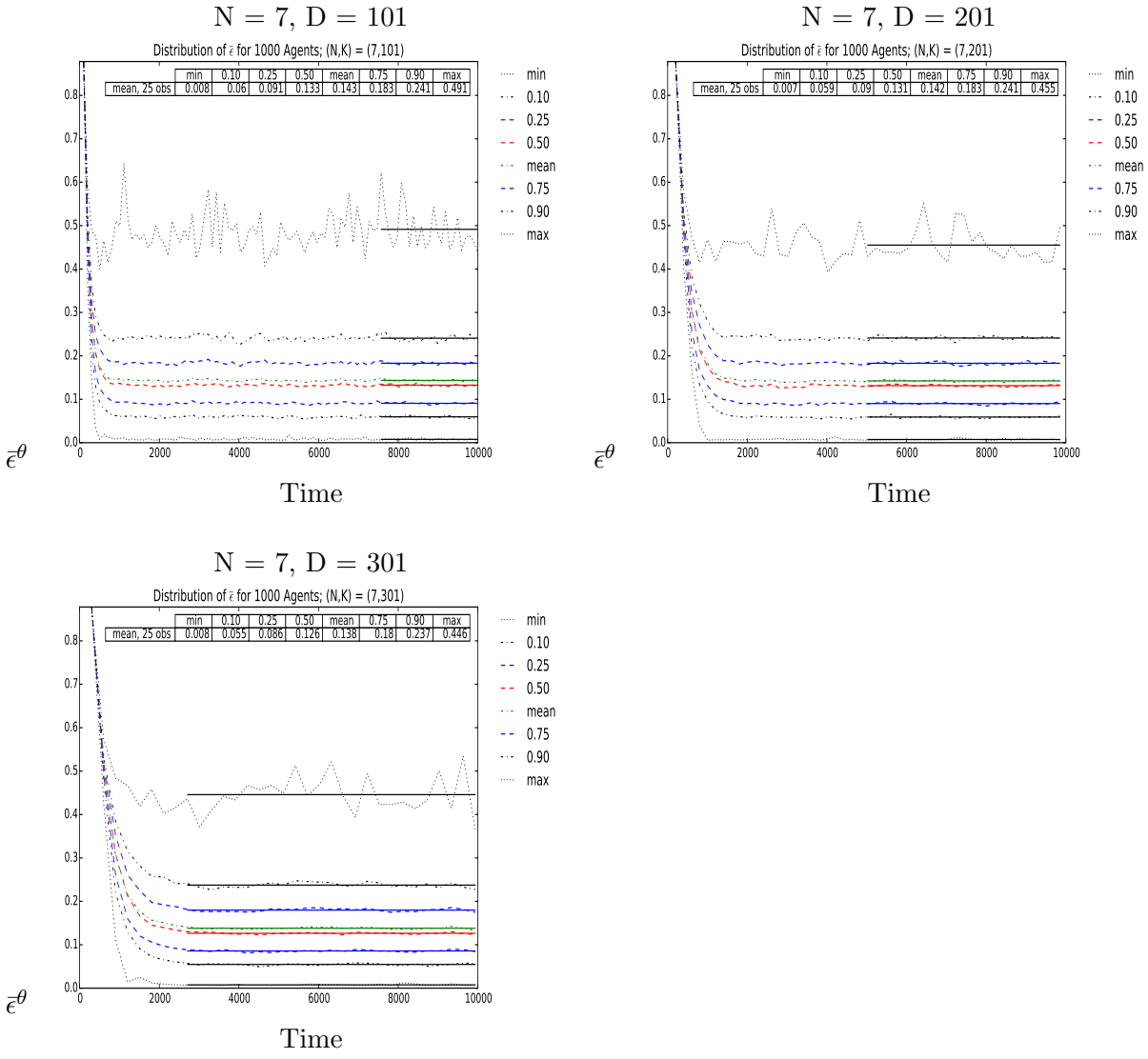


Figure 2.17: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 7, D = 101, 201, 301

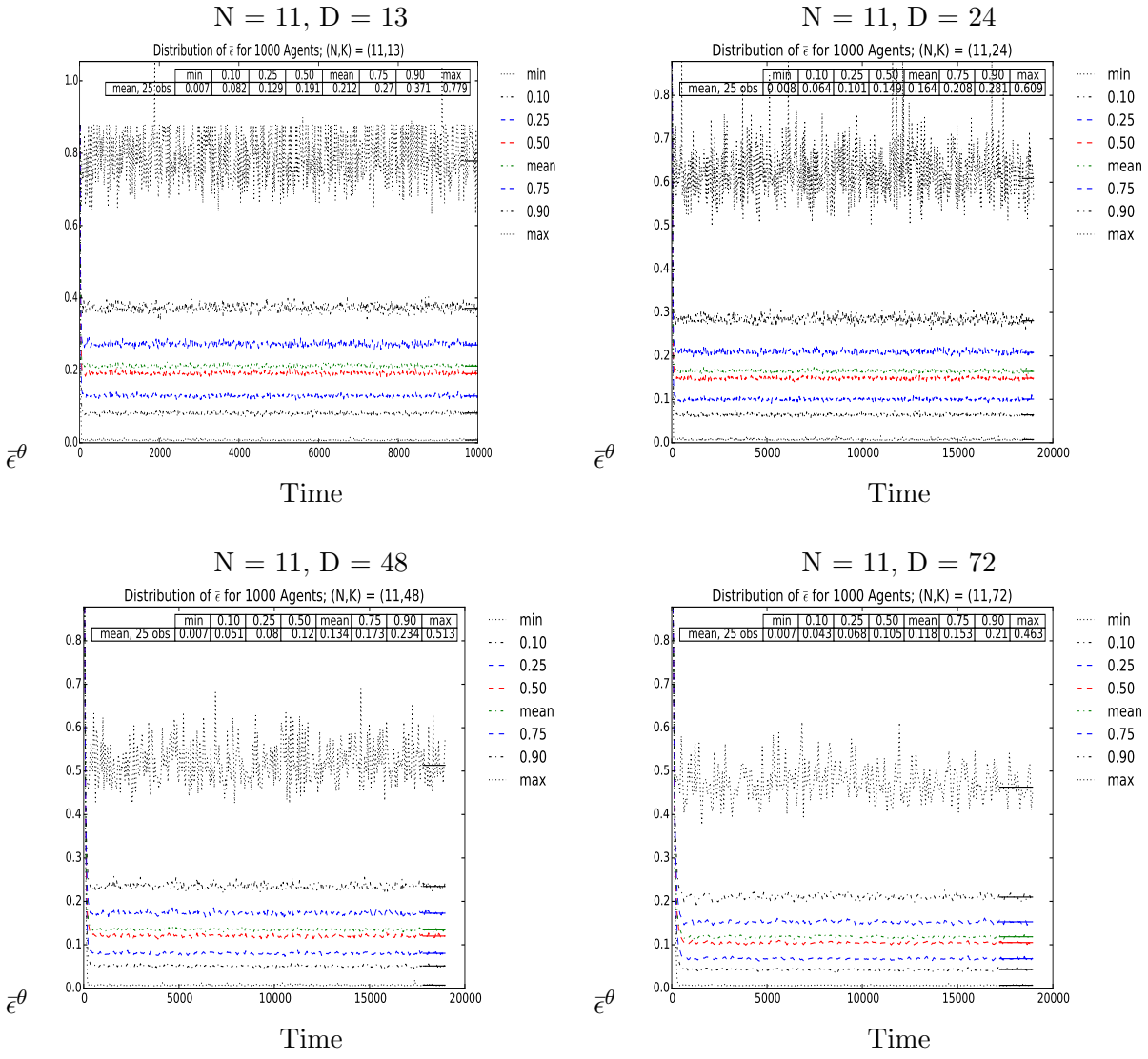


Figure 2.18: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 11, D = 13, 24, 48, 72

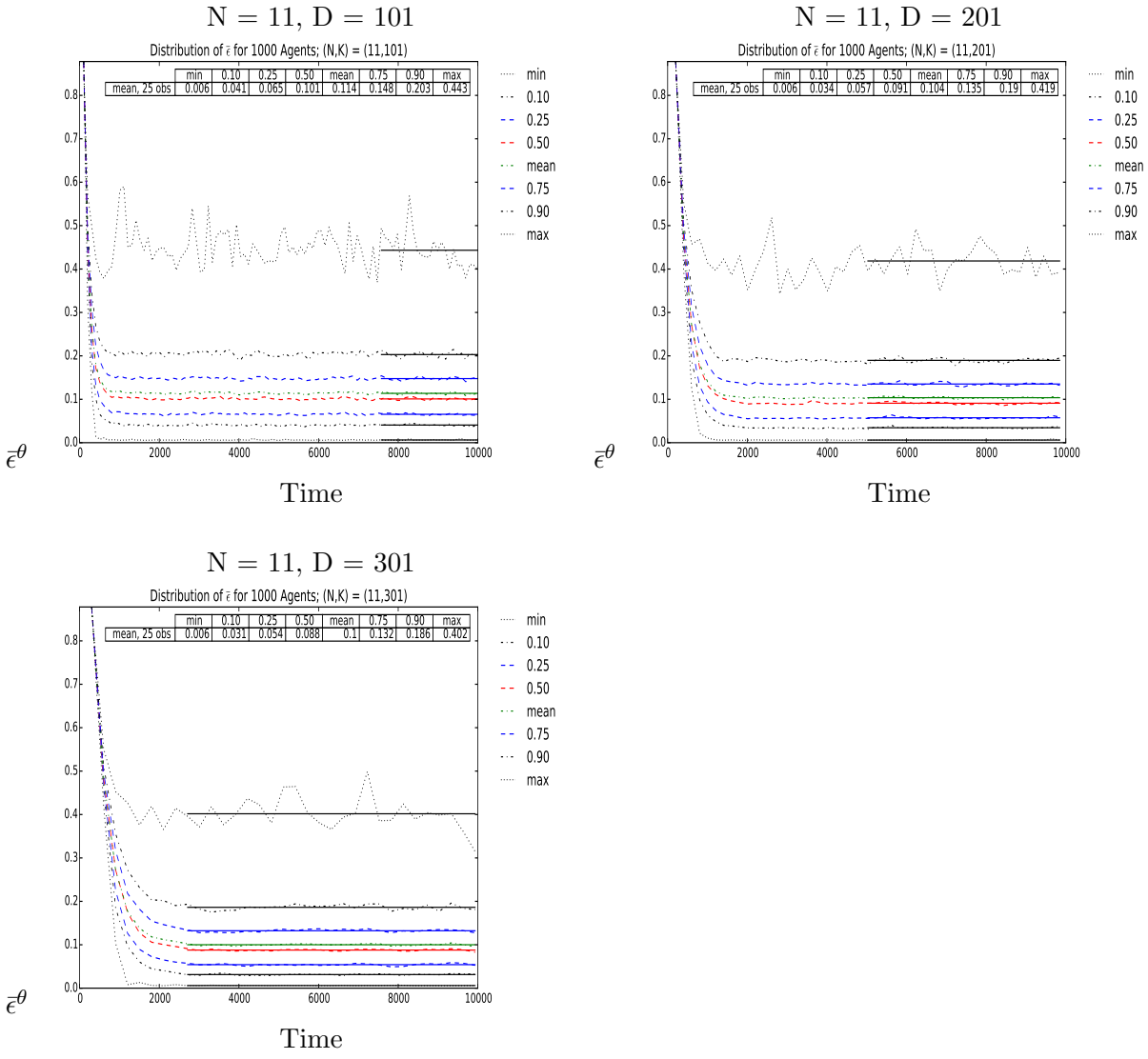


Figure 2.19: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 11, D = 101, 201, 301

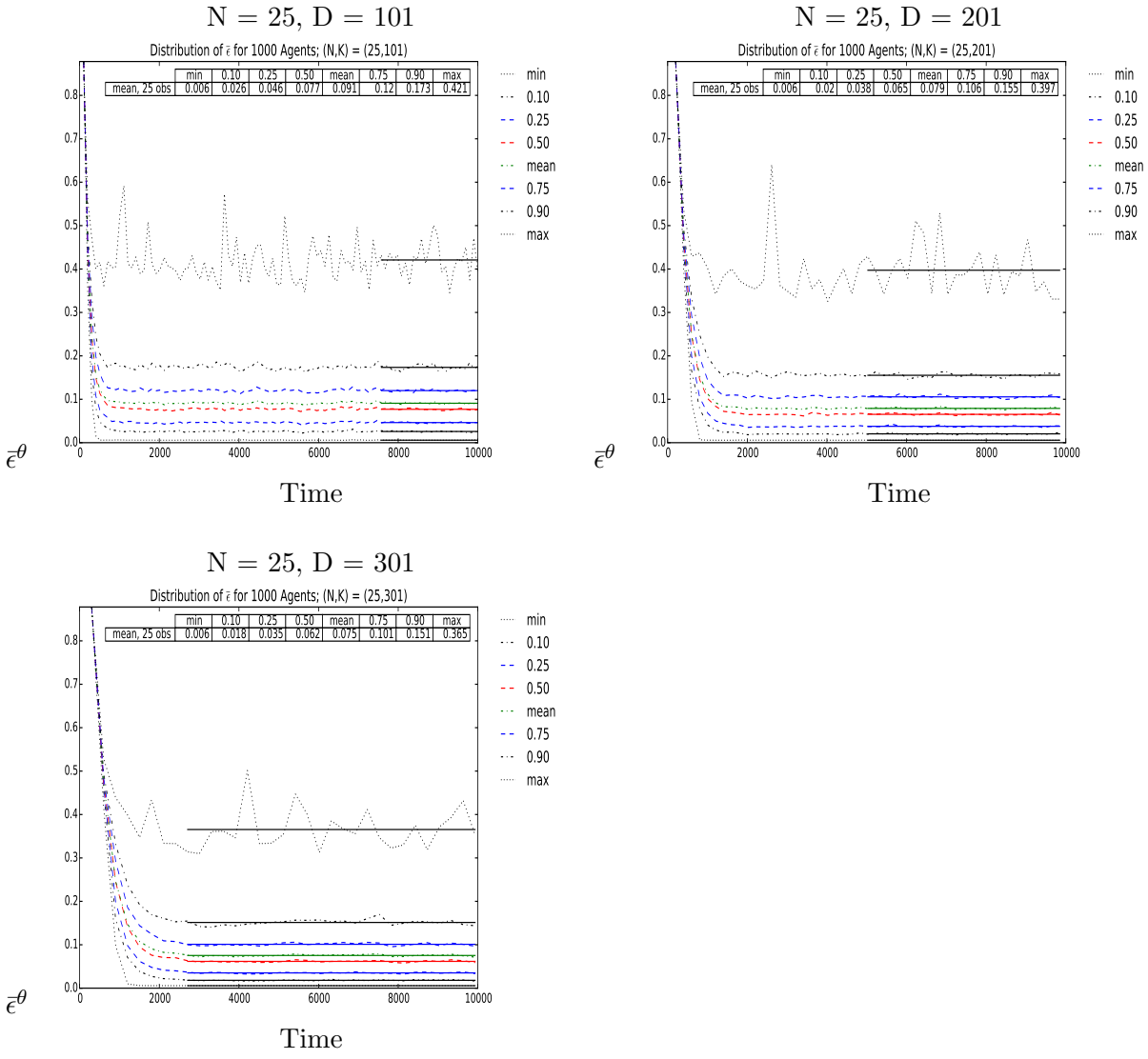


Figure 2.20: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 25, D = 101, 201, 301

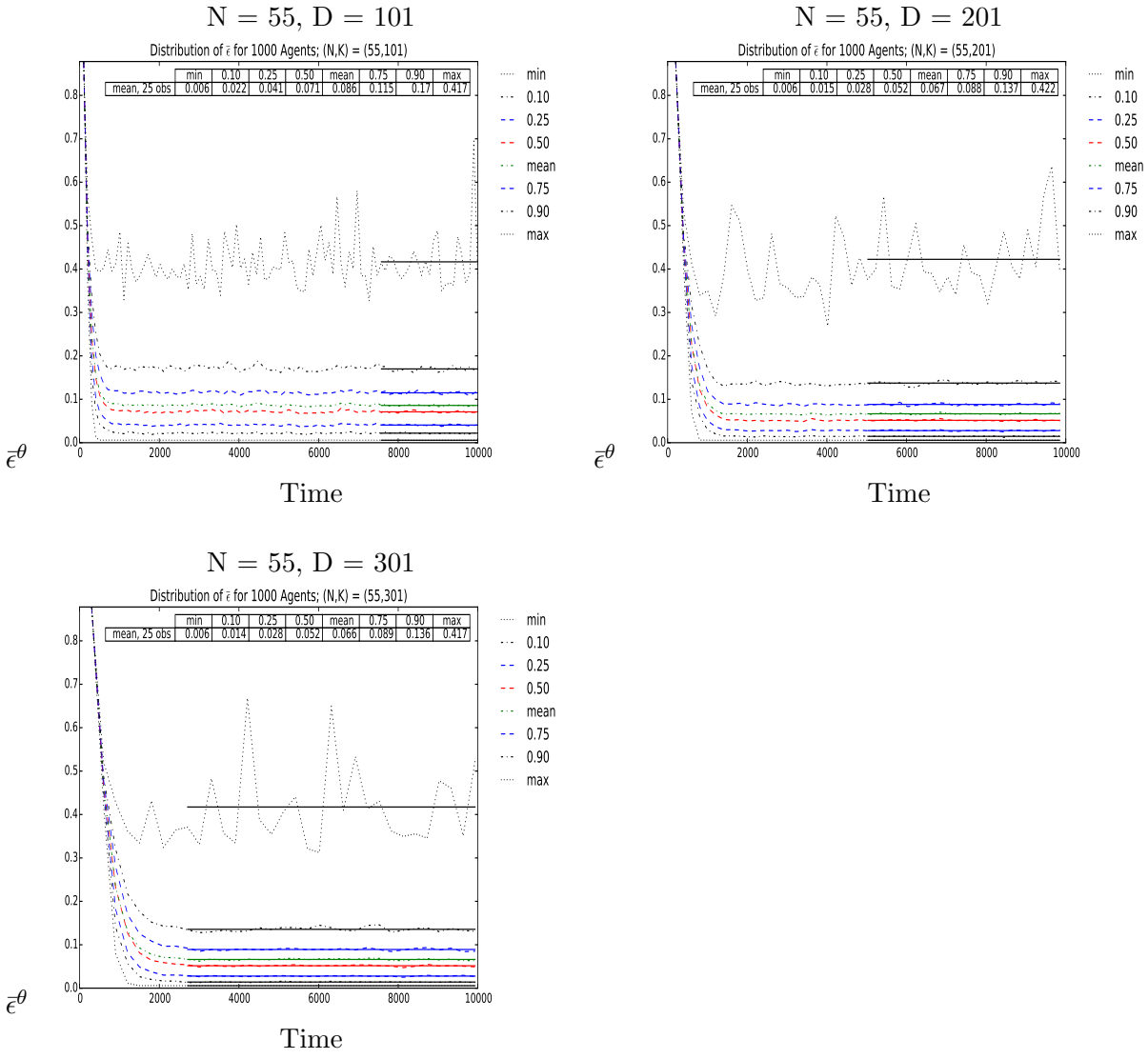


Figure 2.21: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 55, D = 101, 201, 301

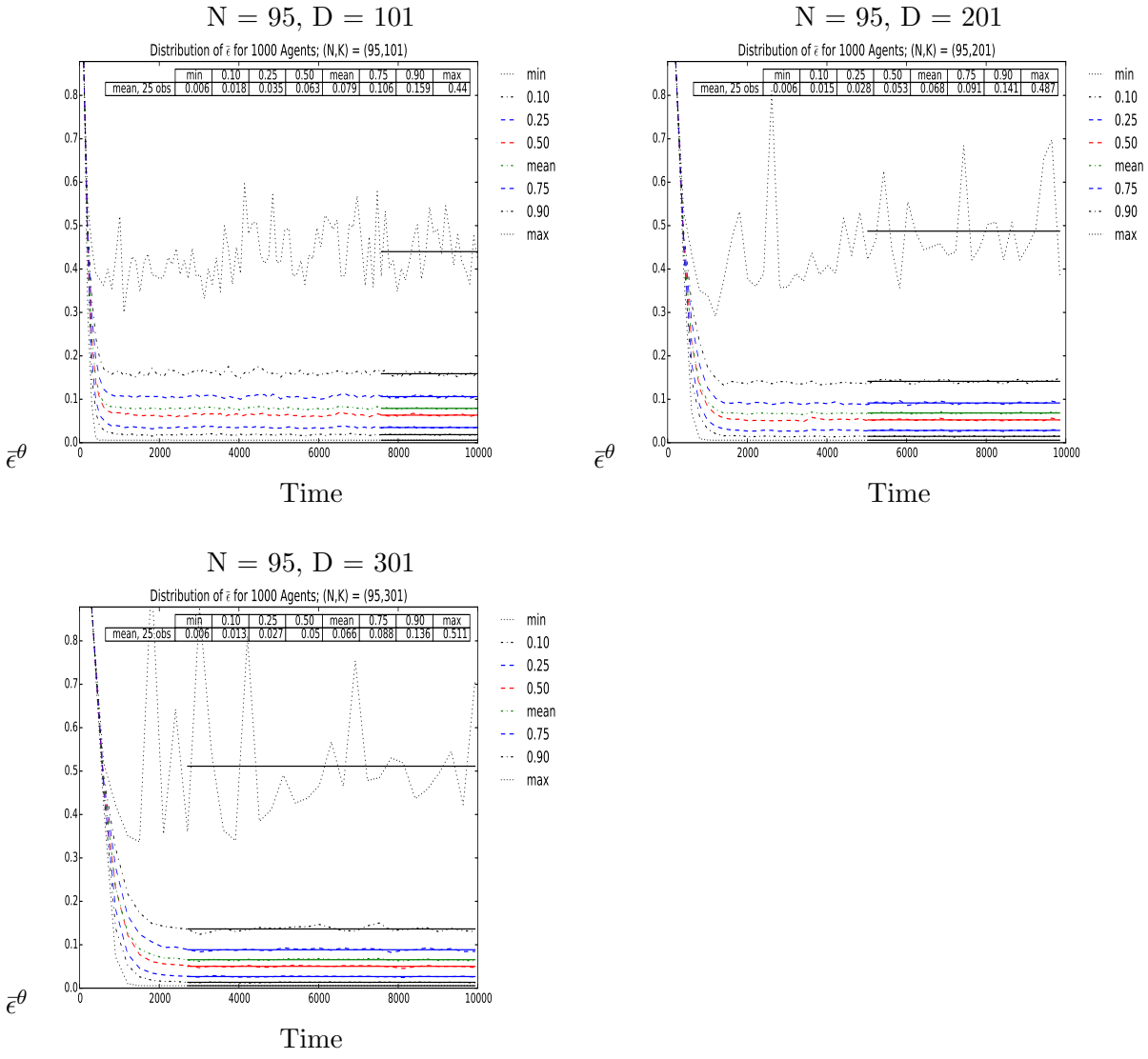


Figure 2.22: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 95, D = 101, 201, 301

Each plot in Figures (2.12) through (2.22) display the average of the distributional characteristics – min, median, mean , max, and 10^{th}, 25^{th}, 75^{th}, and 90^{th} percentiles – for the final 25 periods. There are displayed in each plot, and additionally are recored in agonizing detail in Tables (2.3) and (2.4).

Somewhat easier on the eyes, the mean, median, 90^{th} and 10^{th} percentiles rows from Tables (2.3) and (2.4) are displayed in Figures (2.23) through (2.26), along with the $90^{th} - 10^{th}$ inter-percentile range, a measure of distribution dispersion, displayed in Figure (2.27).

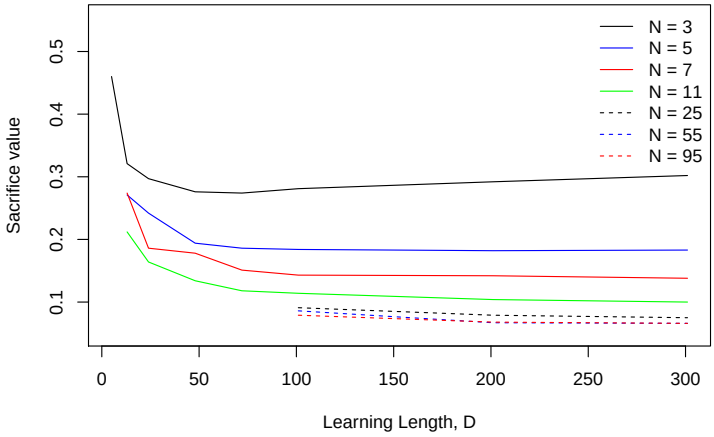


Figure 2.23: Mean of $\bar{\epsilon}^{\theta}$ Distribution for all N x D values

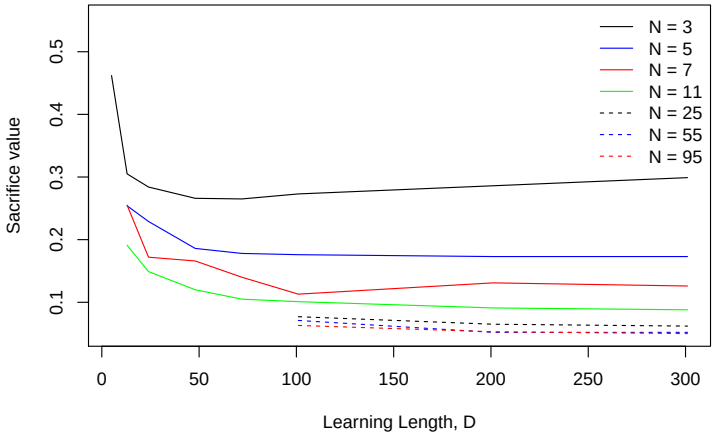


Figure 2.24: Median of $\bar{\epsilon}^{\theta}$ Distribution for all N x D values

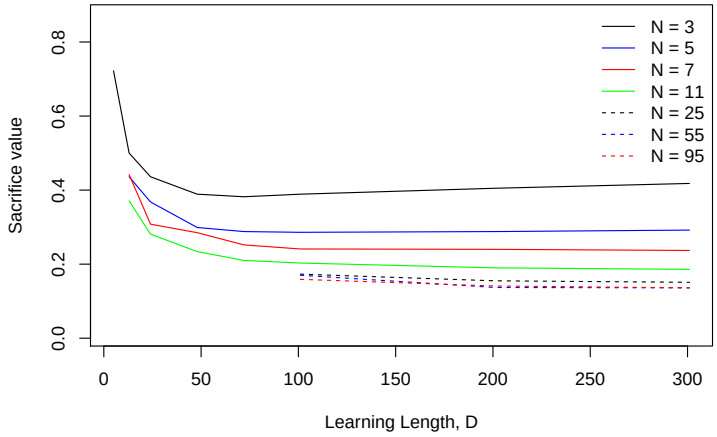


Figure 2.25: 90^{th} Percentile of $\bar{\epsilon}^{\theta}$ Distribution for all N x D values

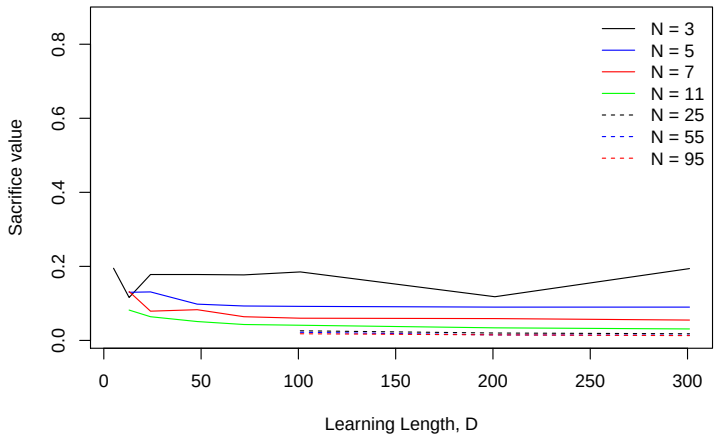


Figure 2.26: 10^{th} Percentile of $\bar{\epsilon}^{\theta}$ Distribution for all N x D values

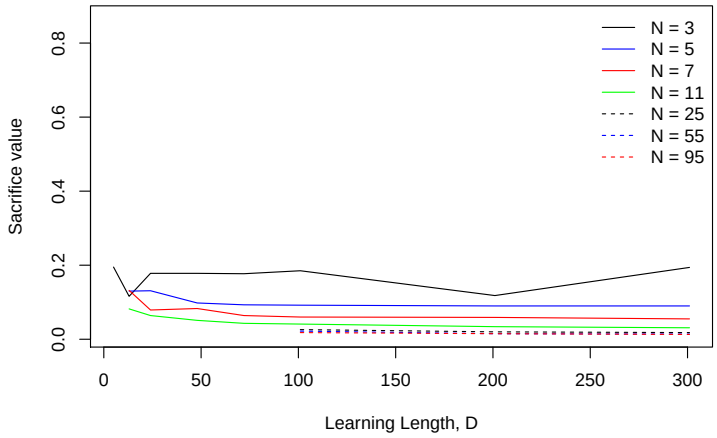


Figure 2.27: $90^{th} - 10^{th}$ Inter-Percentile Range for $\bar{\epsilon}^{\theta}$ Distribution for all N x D values

Table 2.3: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 3,5,7,11, D = 13, 24, 48, 72, 101, 201, 301

N=3	5	13	24	48	72	101	201	301
min	0.008	0.016	0.041	0.06	0.056	0.065	0.061	0.069
0.1	0.195	0.116	0.178	0.178	0.177	0.185	0.118	0.194
0.25	0.313	0.227	0.225	0.218	0.218	0.225	0.232	0.238
0.5	0.462	0.305	0.284	0.266	0.265	0.273	0.286	0.299
mean	0.46	0.321	0.297	0.276	0.274	0.281	0.292	0.302
0.75	0.608	0.400	0.355	0.322	0.032	0.328	0.346	0.361
0.9	0.722	0.500	0.436	0.389	0.382	0.389	0.405	0.418
max	0.91	0.797	0.735	0.646	0.611	0.006	0.588	0.576
N=5	5	13	24	48	72	101	201	301
min	-	0.011	0.02	0.013	0.013	0.015	0.015	0.012
0.1	-	0.13	0.131	0.098	0.093	0.092	0.09	0.09
0.25	-	0.184	0.176	0.138	0.132	0.013	0.127	0.126
0.5	-	0.254	0.229	0.186	0.178	0.176	0.173	0.173
mean	-	0.271	0.242	0.194	0.186	0.184	0.182	0.183
0.75	-	0.339	0.294	0.239	0.231	0.228	0.228	0.232
0.9	-	0.436	0.368	0.299	0.288	0.286	0.288	0.292
max	-	0.777	0.685	0.059	0.539	0.517	0.501	0.494
N=7	5	13	24	48	72	101	201	301
min	-	0.015	0.009	0.011	0.008	0.008	0.007	0.008
0.1	-	0.132	0.079	0.083	0.064	0.06	0.059	0.055
0.25	-	0.187	0.12	0.12	0.096	0.091	0.09	0.086
0.5	-	0.255	0.172	0.166	0.14	0.113	0.131	0.126
mean	-	0.274	0.186	0.178	0.151	0.143	0.142	0.138
0.75	-	0.342	0.235	0.223	0.192	0.183	0.183	0.18
0.9	-	0.442	0.308	0.285	0.252	0.241	0.24	0.237
max	-	0.786	0.624	0.554	0.497	0.491	0.455	0.446
N=11	5	13	24	48	72	101	201	301
min	-	0.007	0.008	0.007	0.007	0.006	0.006	0.006
0.1	-	0.082	0.064	0.051	0.043	0.041	0.034	0.031
0.25	-	0.129	0.101	0.08	0.068	0.065	0.057	0.054
0.5	-	0.191	0.149	0.12	0.105	0.101	0.091	0.088
mean	-	0.212	0.164	0.134	0.118	0.114	0.104	0.1
0.75	-	0.27	0.208	0.173	0.153	0.148	0.135	0.132
0.9	-	0.371	0.281	0.234	0.21	0.203	0.19	0.186
max	-	0.779	0.609	0.513	0.463	0.443	0.419	0.402

Table 2.4: Distribution of $\bar{\epsilon}^{\theta}$ Over Time, N = 25,55,95, D = 101, 201, 301

N=25	5	13	24	48	72	101	201	301
min	-	-	-	-	-	0.006	0.066	0.006
0.1	-	-	-	-	-	0.026	0.02	0.018
0.25	-	-	-	-	-	0.046	0.038	0.035
0.5	-	-	-	-	-	0.077	0.065	0.062
mean	-	-	-	-	-	0.091	0.079	0.075
0.75	-	-	-	-	-	0.12	0.106	0.101
0.9	-	-	-	-	-	0.173	0.155	0.151
max	-	-	-	-	-	0.421	0.397	0.365
N=55	5	13	24	48	72	101	201	301
min	-	-	-	-	-	0.006	0.006	0.006
0.1	-	-	-	-	-	0.022	0.015	0.014
0.25	-	-	-	-	-	0.041	0.028	0.028
0.5	-	-	-	-	-	0.071	0.052	0.052
mean	-	-	-	-	-	0.086	0.067	0.066
0.75	-	-	-	-	-	0.115	0.088	0.089
0.9	-	-	-	-	-	0.17	0.137	0.136
max	-	-	-	-	-	0.417	0.422	0.417
N=95	5	13	24	48	72	101	201	301
min	-	-	-	-	-	0.006	0.006	0.006
0.1	-	-	-	-	-	0.018	0.015	0.013
0.25	-	-	-	-	-	0.035	0.028	0.027
0.5	-	-	-	-	-	0.063	0.53	0.05
mean	-	-	-	-	-	0.079	0.068	0.066
0.75	-	-	-	-	-	0.106	0.091	0.088
0.9	-	-	-	-	-	0.159	0.141	0.136
max	-	-	-	-	-	0.44	0.487	0.511

There are a number of observations one can take from Figures (2.12) through (2.22), Tables (2.3) and (2.4), and particularly Figures (2.23) through (2.26).

Agents converge to a fairly stable distribution of sacrifice values relatively quickly, well within 5-10 episodes of beginning with a spendthrift "consumer everything" consumption function. This can be difficult to see for low-D Figures, such as Figure (2.12), but can be seen much more clearly for higher-D values such as Figure (2.13).

To discuss the effectiveness of regret learning, I focus on the mean and median sacrifice values, averaged over the last 25 learning episodes from each simulation.[30] While the mean and median values can be observed in the full distribution plots or tables, these summary values are most easily seen in Figures (2.23) and (2.24). Here it is readily clear that for a fixed partition N value, the mean and median sacrifice value are nearly flat past a learning-length of roughly 75-100 periods – this can literally be seen as the flat portions of the blue, green, and red lines in each plot.[31] In addition, it can also be seen that there are smaller and smaller improvements in learning when increasing N for a fixed D. For example, for $D = 100$, there is a large improvement in moving from $N = 3$ to $N = 5$, a slightly smaller improvement for $N = 5$ to $N = 7$, somewhat smaller again from $N = 7$ to $N = 11$, and even smaller again from $N = 11$ to $N = 25$, a the largest increase in N for the smallest improvement thus seen. From $N = 25$ to $N = 55$, a nearly doubling of the number of partitions, the improvement is almost indistinguishable. The same is true for the next near-doubling of N-values, from $N = 55$ to $N = 95$.

For the highest N, D combinations, for example $N = 95, D = 301$, is to possible to push the median sacrifice value to 5%. This can be clearly seen in Table (2.4). Even the 90^{th} percentile achieves a sacrifice value of ˜15% across all episode lengths. This is a more than 80% decrease from the original sacrifice value of 90% under the spendthrift consumption function.

[30]Recall that a learning episode is D periods long – so for example, 25 episodes of $D = 25$ periods each totals to 625 periods.

[31]The slight upward trend for the $N = 3$ plot is unusual, and there is not yet a good explanation for this.

2.5.2 Individual-Level Results

I now turn to the experience of the individual. An important question is whether or not an agent's position in the distribution of sacrifice values is persistent. That is, does an agent learn a relatively bad rule and "get stuck there," or vice versa, learn a relatively good rule and forever reap the associated rewards?

The answer is no – once past the first few episodes, agents move freely throughout their distribution. There are a number of was to examine this: directly estimate an AR process on the time series of each agent's position in the distribution of sacrifice values the persistence in an AR process, or examine the distribution of consecutive periods spent in deciles of the sacrifice distribution, or look at the distribution of longest streaks for all agents. All measurements, however, indicate that there is very little persistence of an agent's *relative* position in the distribution of sacrifice values over time. In fact the distribution of sacrifice values over time appears to be an excellent description of what the agent can expect one period ahead. For example, an agent with $N = 5$ and $D = 24$ has a roughly 10% chance of attaining a sacrifice value ≤ 0.13 in a following episode, and a roughly 90% change of attaining a consumption rule with a sacrifice value ≤ 0.3.

The reason for the lack of persistence in agent experience is straightforward. Regret learning takes a stark approach to all information prior to the learning episode: it is forgotten completely. If the regret learning algorithm described in Section (2.4.6) (and requisite foundational sections) is examined closely, it can be seen that no information is used from prior episodes when forming both the value function estimate, the regret choices, or the regret-minimizing consumption function. Importantly, the *prior* value functions learned from previous periods are forgotten entirely. This is also departure from the algorithms for policy iteration and optimistic policy iteration, specifically in the policy evaluation step.[32]

[32]It is not, however, complete departure, because much of the mechanism by which policy iteration is shown to converge relies on the monotonicity of T and T_μ in transforming v. This monotonicity is largely preserved in the parallel approximating steps in regret learning, particularly the single-stream estimation of v discussed in Section (2.4.3). I have yet to show the monotonicity of policy improvement step for regret learning; this is currently underway as as part of future work.

The reason for this omission is straightforward: if agents are to remember all past information, one must take a stance on how this is incorporated into agent behavior. The single-stream Monte Carlo estimator has no simple way to incorporate previous value functions. A number of different potential approaches each raise important methodological questions. If one were to simply attempt to average in previous experience, even aside from issues raised by the shifting partitions, one must take a stance on whether much earlier information should be treated differently than very recent information – that is, one needs to take a stance on the "gains" which may be used.[33] Having agents forget information completely when starting each new episode is a clean, simple first step in exploring the behavior of regret learning.

The upside of this stark stance on previous information is that interesting dynamics in agent learning are clearly highlighted. For example, Figures (2.29) through (2.59) display a single run for an individual agent with $N = 11$ partitions and $D = 25$ learning periods.[34] I will use this sequence of plots trace out a typical path through the consumption space. The plots are paired, a value function plot followed by a consumption function plot. The value function plot displays two *theoretical* value functions: the true optimal value function in black and in red, the *theoretical* value function associated with the consumption function c^{θ_k} employed by the agent in episode k. The purpose of this plot is to give us an idea about how far, in theoretical terms, the agent is from the true optimal solution; this is crystallized by the sacrifice value for the consumption function c^{θ_k} displayed in the legend of this plot.

[33]See LeBaron (2012) for an excellent examination of related questions in agent-based models generally, and see Evans and Honkapohja (2001) for a broad discussion of the topic in macroeconomic learning.

[34]To avoid biased selection of results, this agent was accepted as the first result which emerged from randomly selecting the starting seed for the random number generator which produced this agent's shocks.

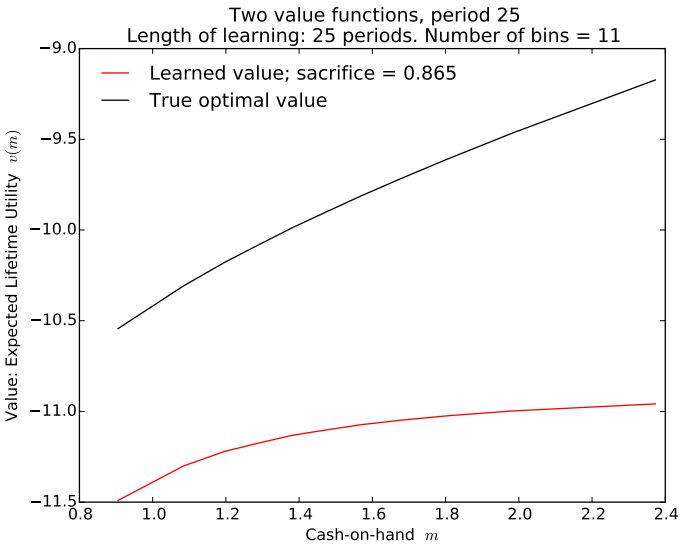


Figure 2.28: Welfare Cost

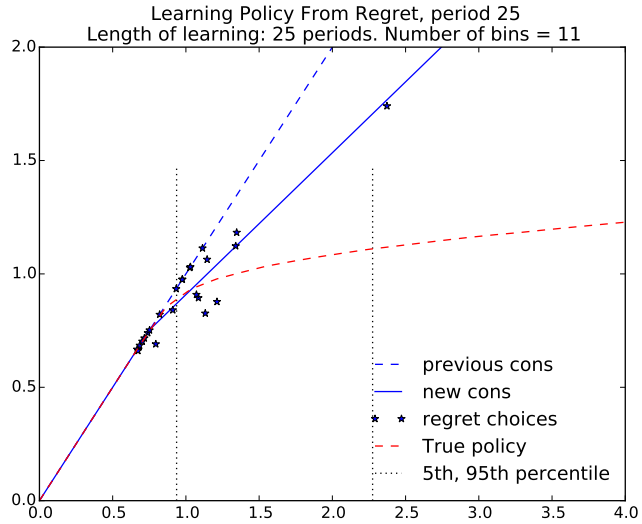


Figure 2.29: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

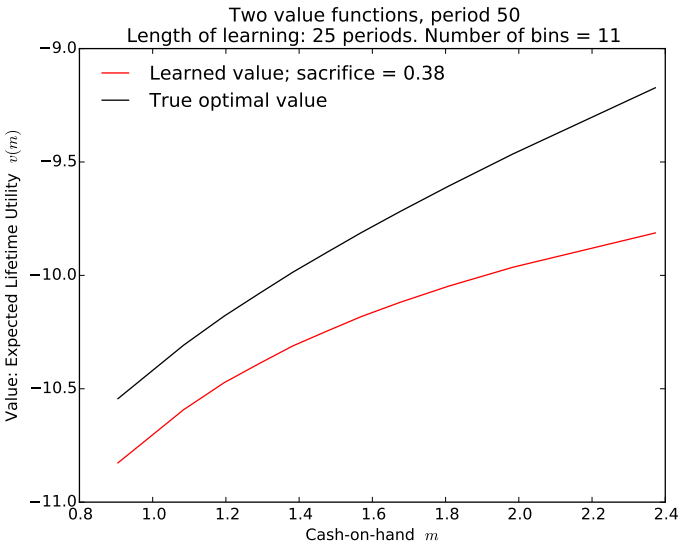


Figure 2.30: Welfare Cost

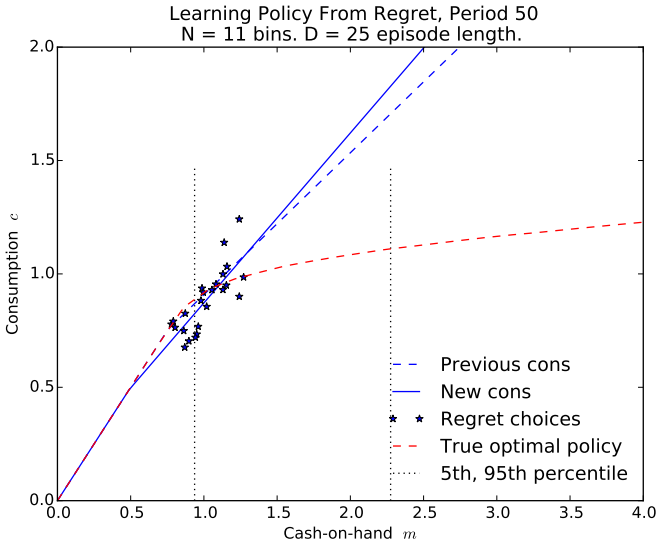


Figure 2.31: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

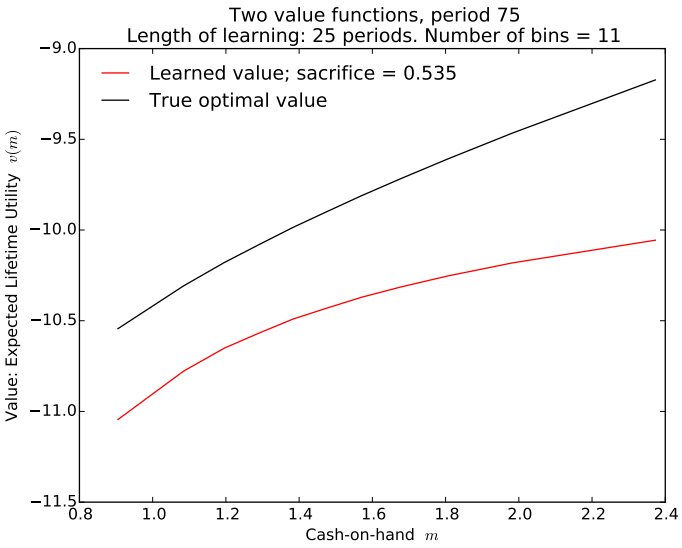


Figure 2.32: Welfare Cost

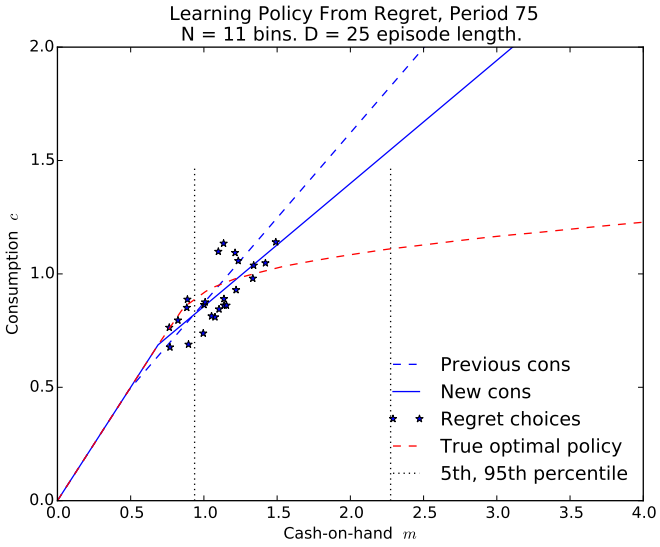


Figure 2.33: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

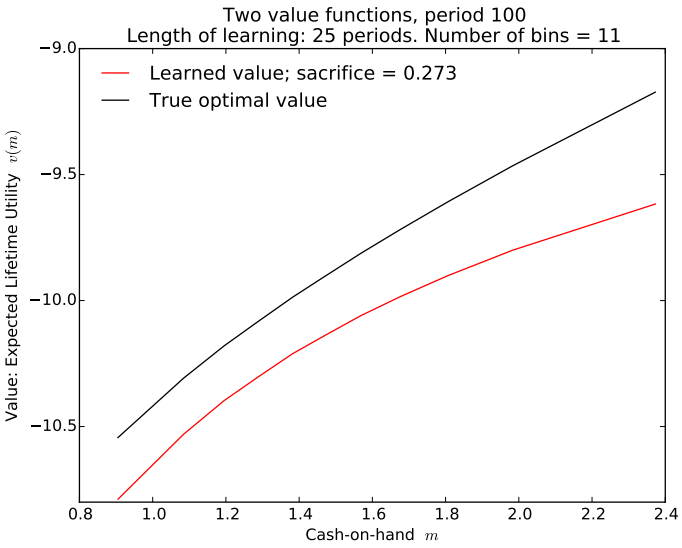


Figure 2.34: Welfare Cost

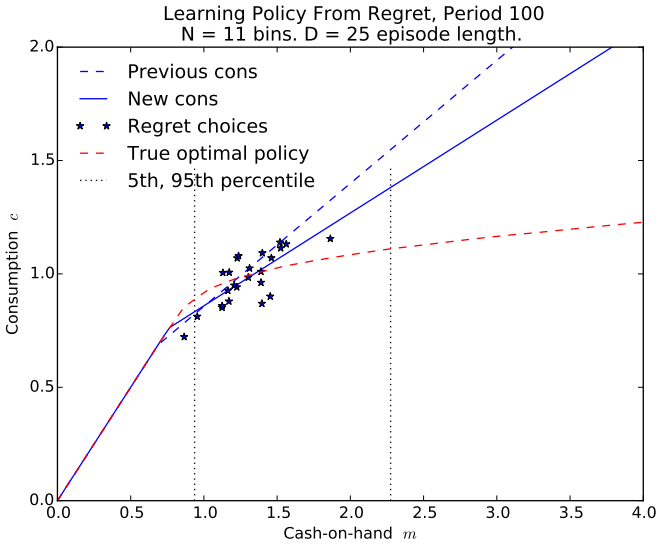


Figure 2.35: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

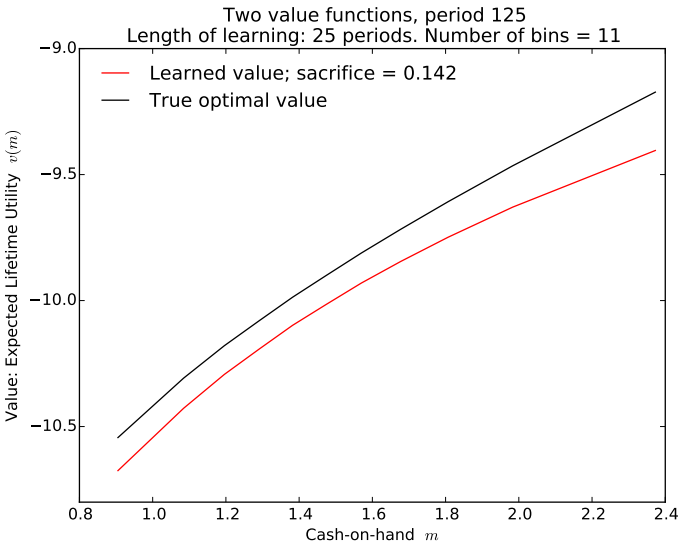


Figure 2.36: Welfare Cost

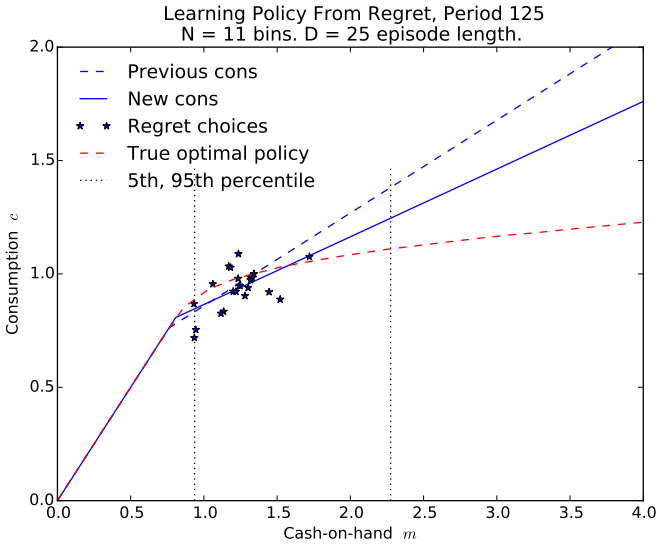


Figure 2.37: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

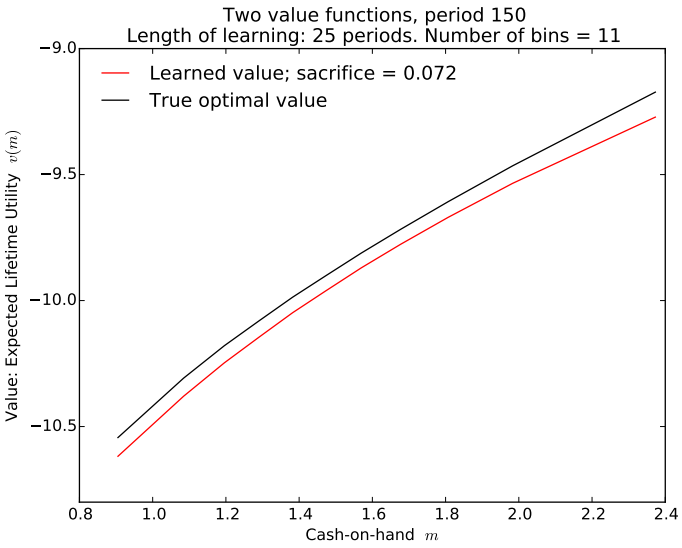


Figure 2.38: Welfare Cost

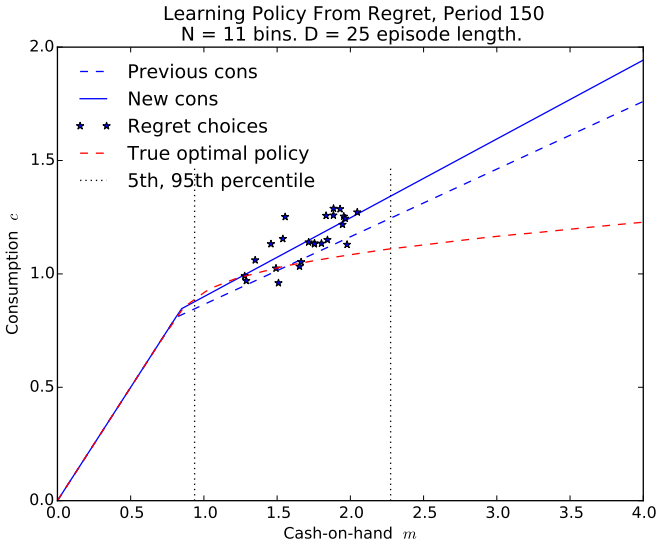


Figure 2.39: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

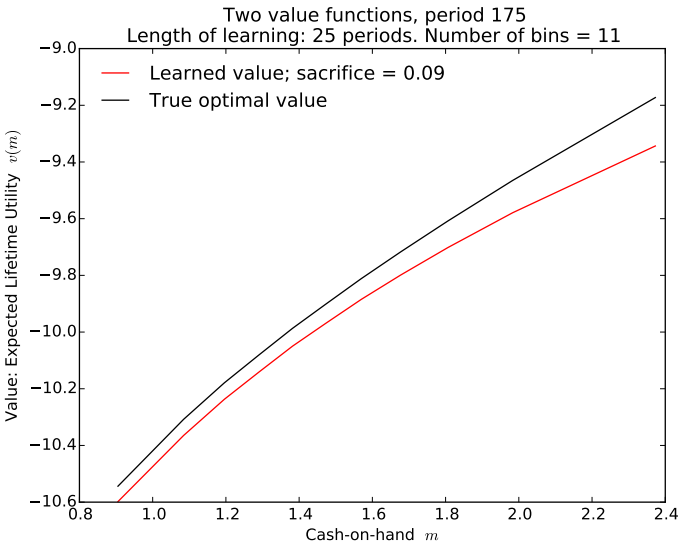


Figure 2.40: Welfare Cost

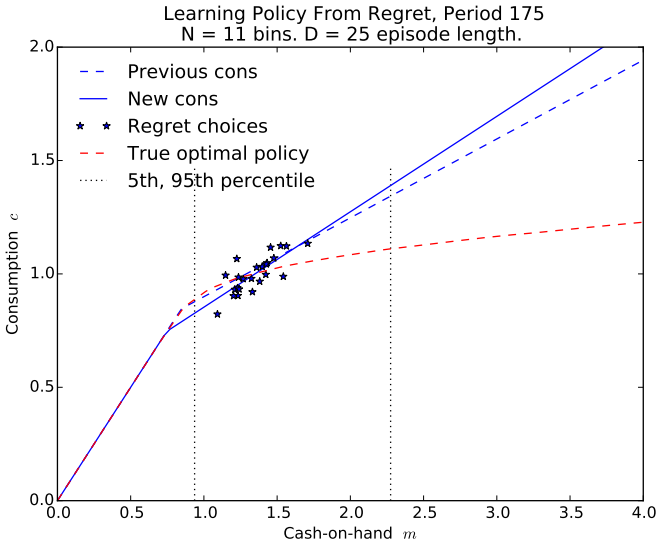


Figure 2.41: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

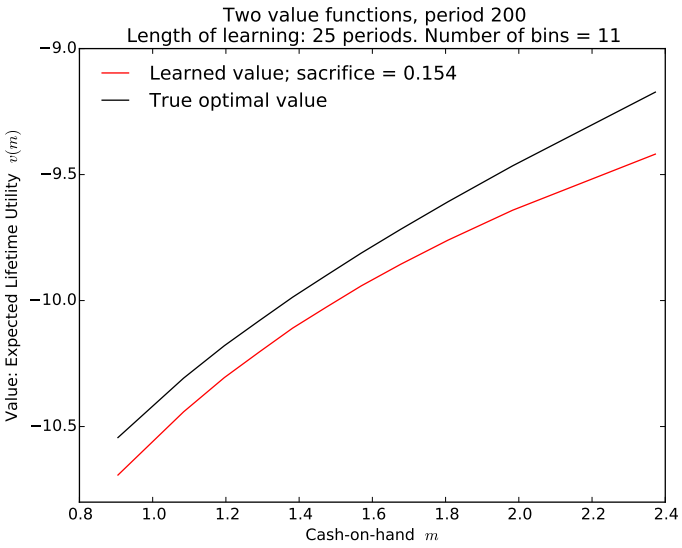


Figure 2.42: Welfare Cost

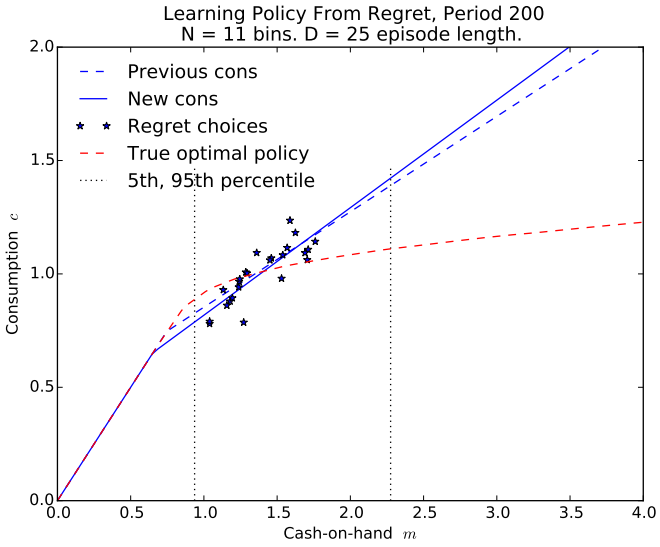


Figure 2.43: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

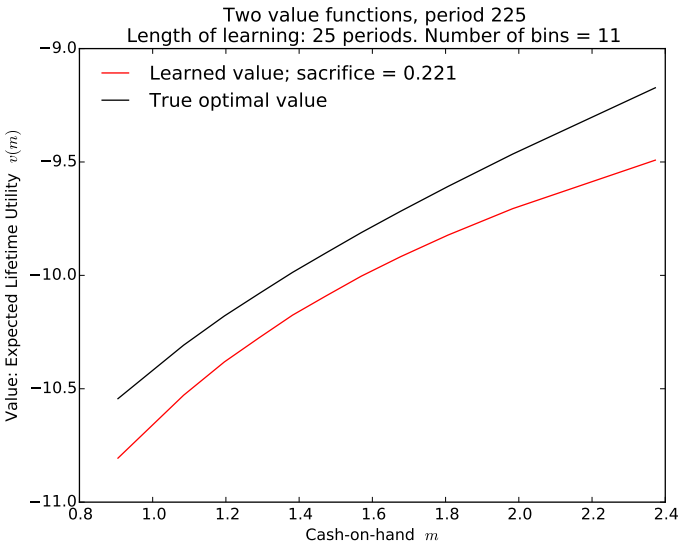


Figure 2.44: Welfare Cost

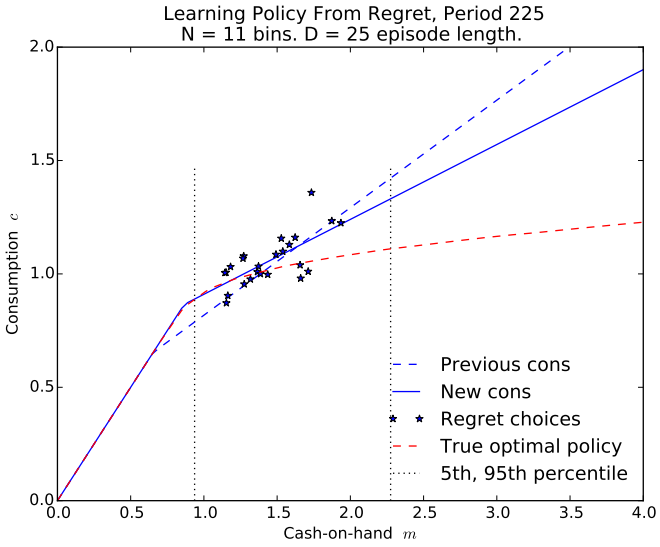


Figure 2.45: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

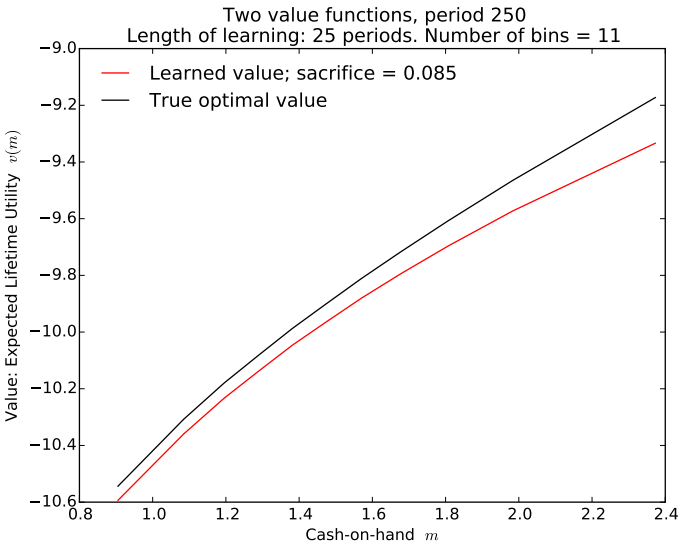


Figure 2.46: Welfare Cost

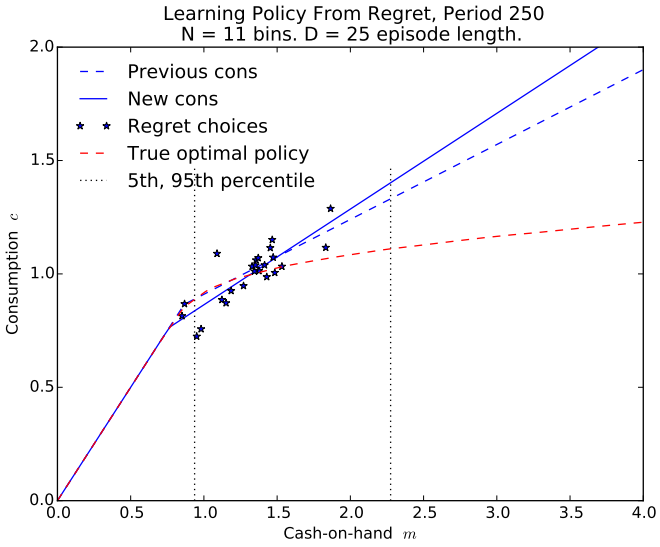


Figure 2.47: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

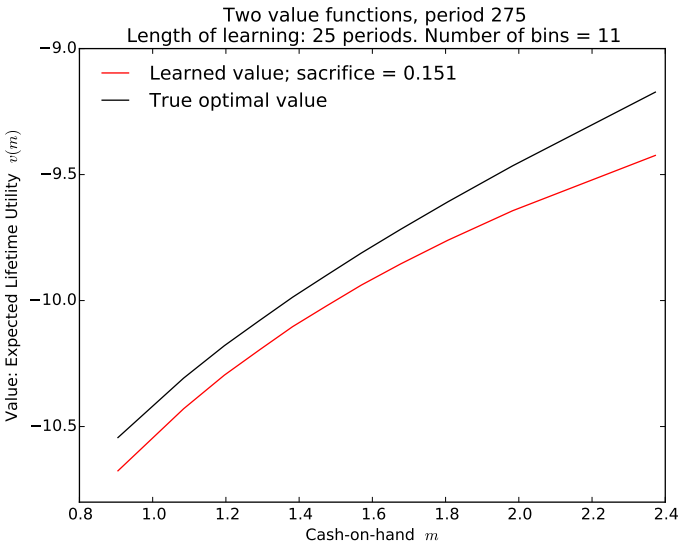


Figure 2.48: Welfare Cost

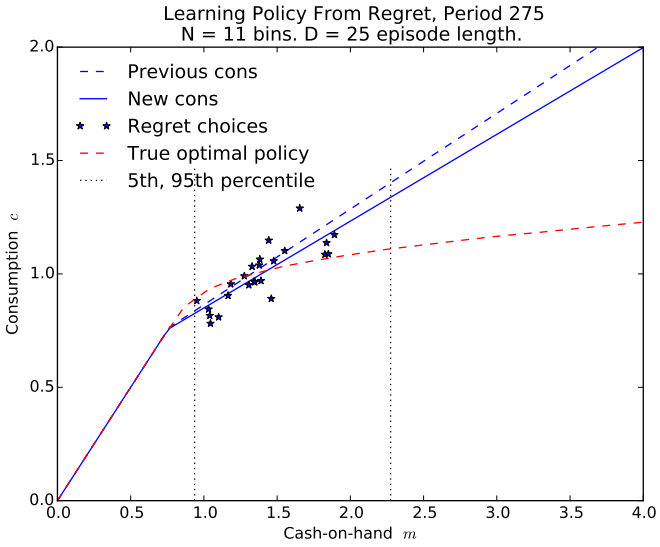


Figure 2.49: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

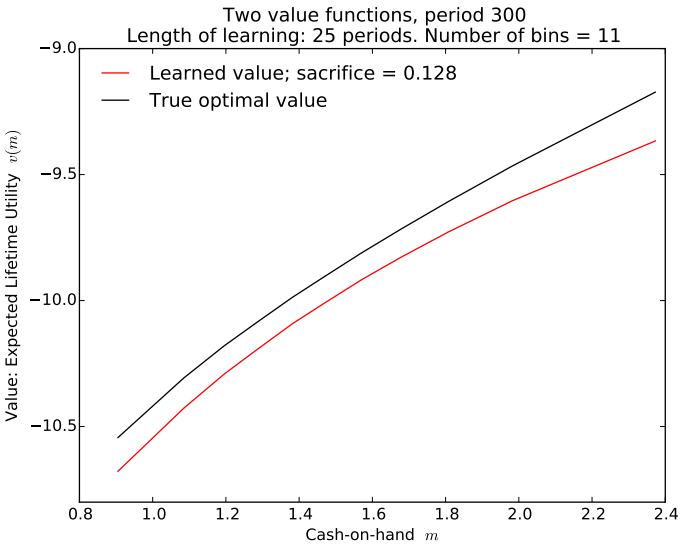


Figure 2.50: Welfare Cost

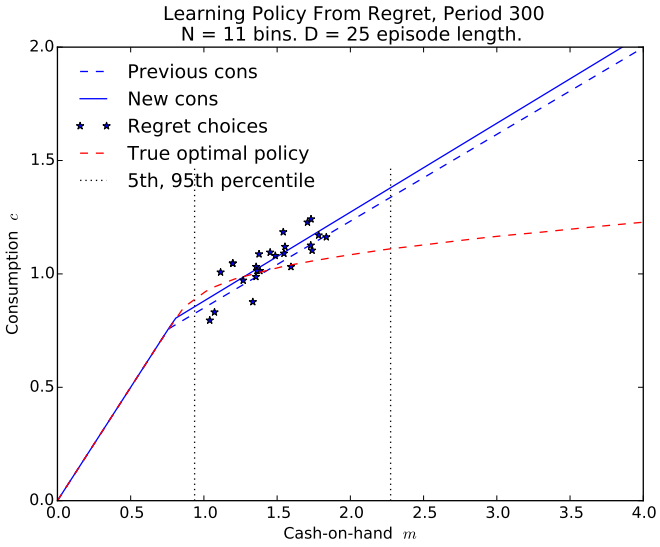


Figure 2.51: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

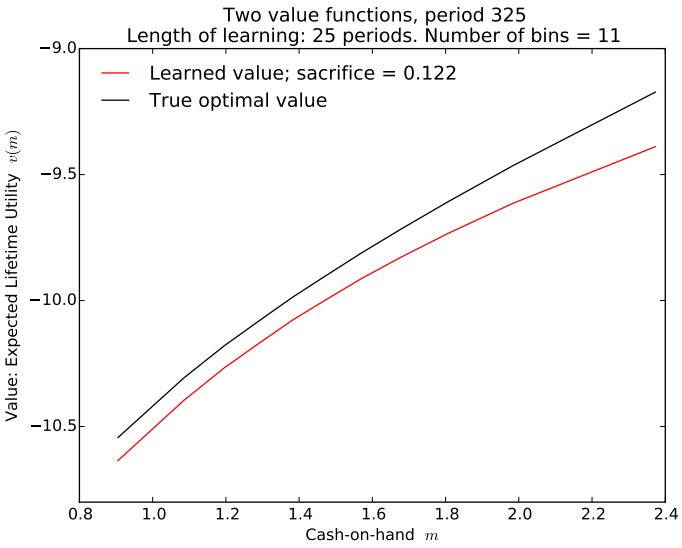


Figure 2.52: Welfare Cost

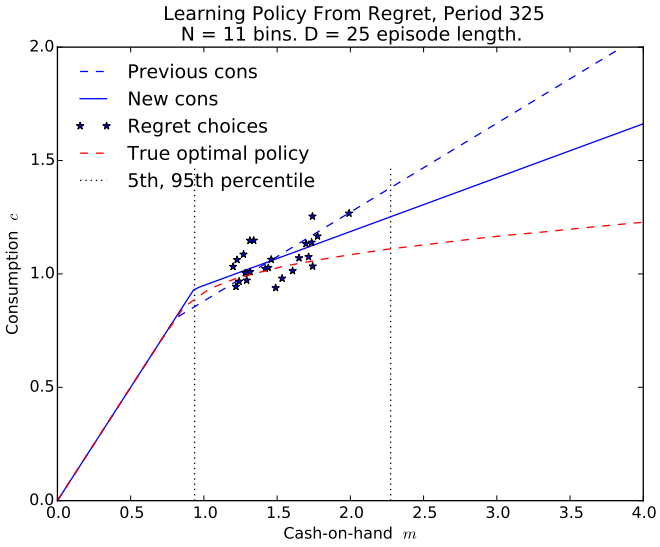


Figure 2.53: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

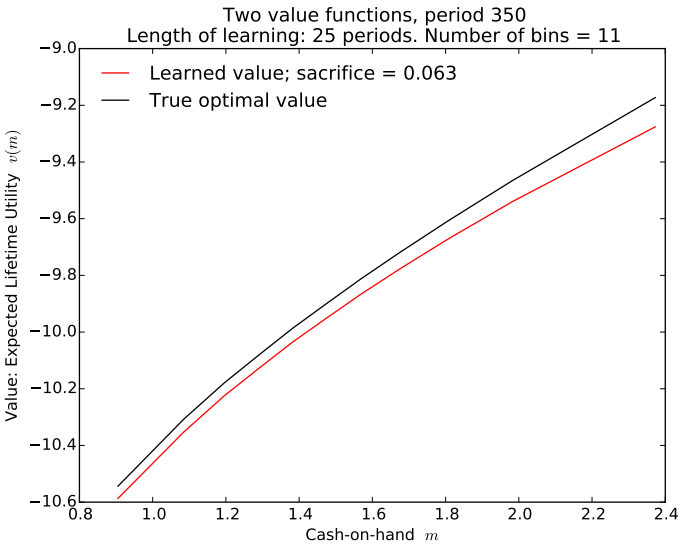


Figure 2.54: Welfare Cost

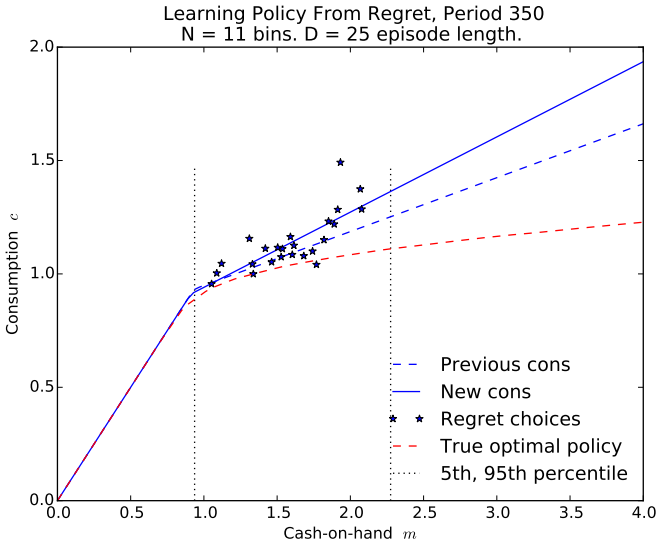


Figure 2.55: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

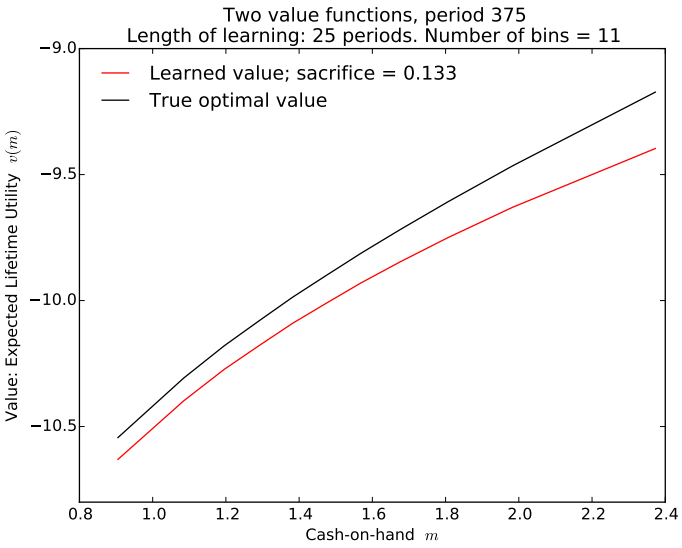


Figure 2.56: Welfare Cost

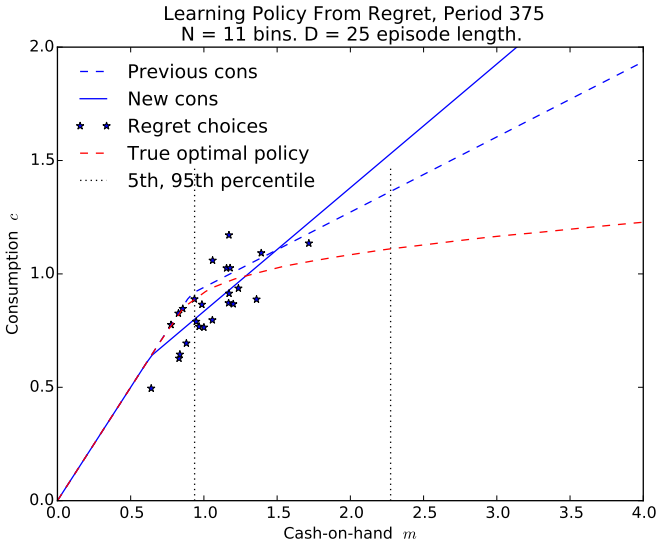


Figure 2.57: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

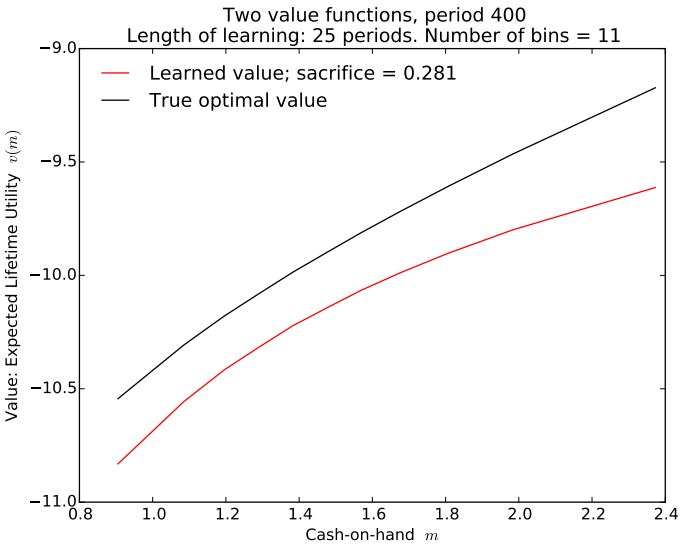


Figure 2.58: Welfare Cost

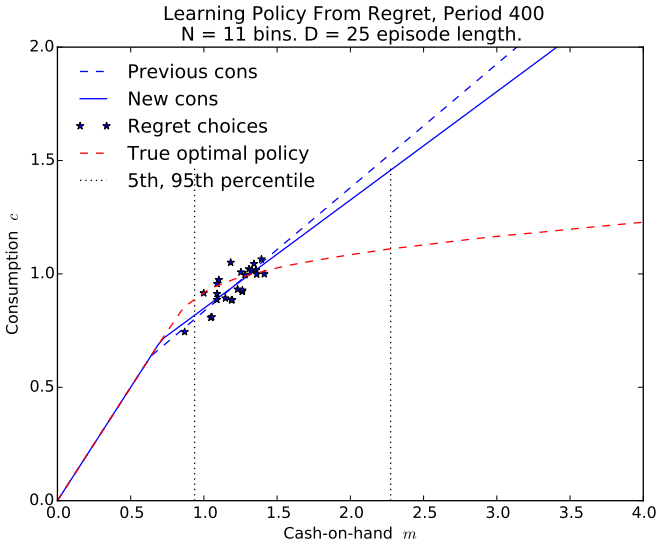


Figure 2.59: Paired Plots: Theoretical Value Functions (Top) and Learned c^{θ} Functions (Bottom)

The consumption plot in each pair of plots displays the theoretical optimal consumption function in a red dashed line, the previous-episode c^{θ_k} function in dashed blue line, the regret choices learned after following that function as blue stars, and finally the updated consumption function $c^{\theta'}_{k+1}$ which minimizes distance between the regret choices. The vertical dashed lines in each consumption plot outline the 5^{th} and 95^{th} percentiles of the ergodic distribution of cash-on-hand , m, following the true optimal rule. These are included to provide a rough indication of how "out of the ordinary" a set of consumption experiences may be with respect to the target optimal policy. When experiences stray outside these boundaries, the agent tends to learn a poor consumption function.

In the plot-pairing of Figure (2.29), the sacrifice value of the initial spendthrift consumption function is quite high, 0.865, or roughly 90% of expected annual income. In the following consumption figure, the dashed blue line is the 45^* line – the initial spendthrift consumption function, c^{θ}_0 . The blue stars represent the regret choices – the choices the agent *wishes* he/she made, after following the spendthrift rule and then the estimated value function is not shown in any plots). The solid blue line is the first improved policy function $c^{\theta'}_1$ which the agent learns from these regret choices – it is already bending towards the optimal policy.

The following paired plots, Figure (2.31), show the theoretical value function and sacrifice value for the previously learned consumption function, $c^{\theta'}_1$. The first value function is already a dramatic improvement over the original spendthrift rule: 0.38 is less than half of the original sacrifice value. Once again, the stars indicate the regret choices which arise from the value function estimated from experience following the $c^{\theta'}_1$ function. Note that the regret values are concentrated around the lower 5^{th} percentile – when the agent minimizes regret through these choices, the resulting consumption function, the solid blue line, visually moves *slightly away* from the true optimal policy. As a result, this episode is less impressive for agent learning. Looking ahead to Figure (2.33) the agent actually retreats slightly in utility terms; the learned function has a sacrifice value of ˜0.54.

There is no need for despair, however – the next updated consumption function, the solid blue line in the bottom plot of Figure (2.33), is moving back on track. Figures (2.35), (2.37) and (2.39) show the agent making solid progress to a consumption function with a 7% sacrifice value.

Continue this examination from (2.39) through the end of the agents recorded experience in Figure (2.59). Conveniently these last figures demonstrate the agent being deceived by experience one final time by a set of regret experiences on the low end of experience – see the consumption update in Figure (2.57) and the subsequent sacrifice value increase in the value portion of Figure (2.59).

2.5.3 Individual-Level Results: Summarized

Finally, the agent's experience as described is summarized in Figure (2.60), which displays the sacrifice values in levels and as a fraction of the spendthrift sacrifice value. It can be seen that the agent experiences improve quickly and then settle into the distribution of sacrifice values appropriate to their D and N parameters, which may be read off of Table (2.3). Note that this agent's uptick in sacrifice value in the final period is essentially right at the 90th percentile cutoff.

2.5.4 Individual-Level Results: Welfare Analysis

One last way to visualize the agent results for different N, D parameter values is to examine what an agent's learning path through the welfare surface depicted in Figure (2.7) looks like. I create two separate agent learning experiences and map their progress through the welfare space (the sacrifice value surface) in Figures (2.61) through (2.66). These plots look "messy" because they only display the experiences of two agents – they are meant only as a stylistic demonstration and not as a description of the distributional characteristics of learning, as the multi-agent simulation above capture.

The first figure demonstrates the two agents learning, one denoted by the connected blue dots, and one denoted by the connected green dots. As in all examples, agents start at the

"consume everything" spendthrift rule on the far right side of the plots where $\bar{m} = \kappa = 1.0$ and learn for 10 episodes. Because episodes will vary in length, the total time will vary, but each plot will display 10 episodes of learning experience for two agents. In the first learning plot, Figure (2.61), the two agents have a sophistication of $N = 5$ and episode length $D = 25$. Their first learning episode takes them directly inside the contour line for $\bar{\epsilon}^{\theta} = 0.4$, and they quickly fall into a rough distribution here. This experience is consistent with that seen in Table (2.3).

The next three Figures, (2.62) through (2.64), demonstrate what learning looks like as we incrementally increase N or D. The central tendency of the learning "clusters" shift closer towards the least-welfare-loss rule as N and D increase.

Figures (2.65) and (2.66) push agent experience to higher extremes, and demonstrate that as sophistication N and episode length D increase, agents tend to learn rules closer and closer to the true optimal rule. The few wild swings seen also illustrate that agents can still be "deceived" by their experience, and learn rules which are not monotonically in the direction of the true optimal rule. Note, for example, the blue agent jumping out of the 0.05 sacrifice ring in Figure (2.64). Importantly, the agents in all plots endogenously avoid the "danger region" in the upper right-hand side of the welfare surface.

2.6 Summary, Conclusion, and Next Steps

The research goal of regret learning is to capture the simplest possible *fully modular* method of learning, which allows agents to converge to a stable, well-defined neighborhood around the true policy in with minimal external information from the modeler. It is meant to be easy to understand and immediately implementable for the economist and agent-based modeler who is familiar with dynamic programming. Furthermore, it has been designed from the ground up to be nearly trivial to take to data – this is a natural result of being closely tied to the dynamic programming framework and use to solve the same problem as are popular in the consumption under uncertainty literature. The Heterogeneous-Agent

Computational toolKit (HACK), presented in Chapter 3, makes the estimation of regret learning almost instantaneous, using the same data and estimation methods (Simulated Method of Moments, SMM, with bootstrapped standard errors) as the traditional framework. The extensive effort presented in the Consumer Problem section aims to make this model compatible with the consumer problem as presented in HACK. The fact that the HACK framework estimates a finite-horizon consumption savings buffer-stock problem does not present a problem; as has been often observed[35] households do not seem to accumulate liquid savings for retirement until roughly 45-50 years old. This leaves at least 20-30 years of data against which the infinite-horizon model above can be estimated without fear of structural mismatch.

There are a number of possible extensions and next steps for regret learning. These include:

- Estimating the model against data to establish a fit for risk aversion, the discount factor, and N and D parameters, if so desired.
- Implementing variations of the regret learning algorithm:
  - regret learning using simpler alternative value estimators, such as nearest neighbor approximations, which allow flexibility in choosing $D < N$ parameters,
  - regret learning in an optimistic policy iteration framework, which would update the value function each time step instead of each episode; in such a case the policy update would still occur each episode but the value function would evolve more quickly, or
  - regret learning which carries the value function forward from one episode to the next.
- Use simulation-based model selection, as discussed in Shalizi (2015), to distinguish between different possible variations of the regret-learning framework as suggested above.

[35] See, for example, Cagetti (2003), Carroll and Samwick (1997), and Gourinchas and Parker (2002).

- Continue to work out the convergence properties of the current model.
- Continue to work out the analytical parallel between regret learning and Q-learning.
- Continue to explore the analytical foundations of regent leaning, including application of non-expansive transformations in an approximate policy iteration framework.
- Construct an empirical "horse race" between various versions of the learning algorithms in the small but growing "learning to optimize" literature.

The original motivation for developing regret learning was to provide agents in Chapter 3 with a robust method of local exploration. This has been archived, and in addition regret learning presents an immediate option to include social learning into the model. Because agents estimate the full distribution and full value function in regret learning, they can communicate their unconditional expectation of the value of using their function, and can thus socially share information as in Chapter 3 with any agent in their population.

Finally, this model is explicitly intended to be used in large-scale agent-based models such as Geanakoplos et al. (2012) or the CRISIS model (Hommes and Iori, 2015).

The core of the CRISIS model is an agent-based model with households who explicit act as Allen and Carroll (2001) consumers, using the same linearized form of the consumption function has I use here, but without the learning element. This is a prime example of the appropriate application of regret learning to agent-based models, and the type of application for which it is very well suited.

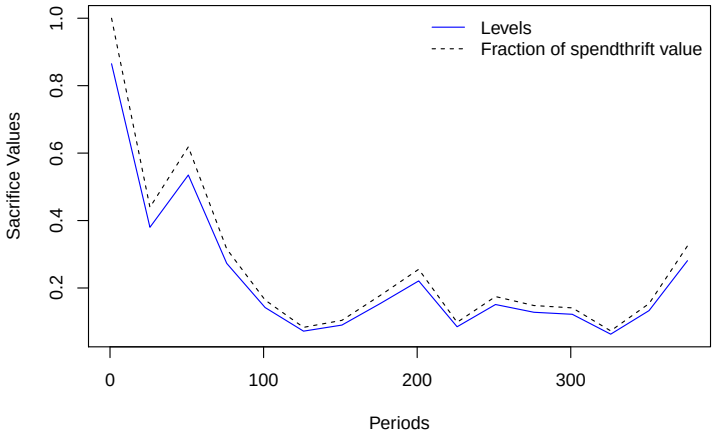


Figure 2.60: Summarized Sacrifice Value Experience

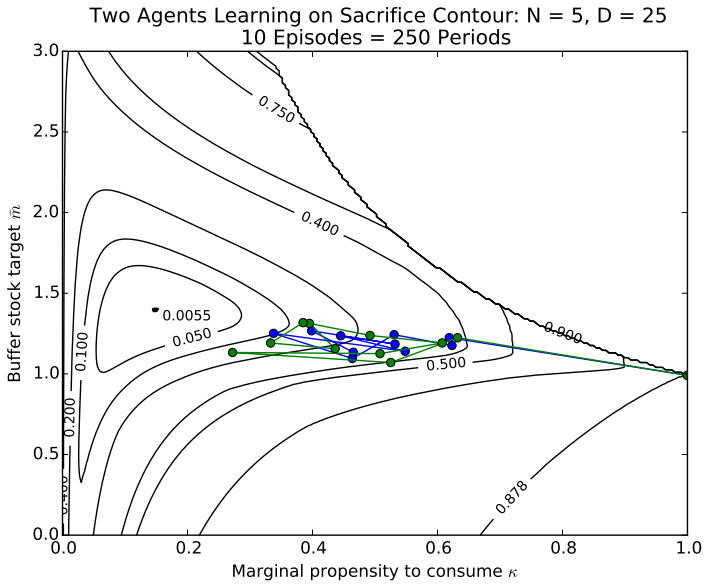


Figure 2.61: Learning from Regret: N = 5, D = 25

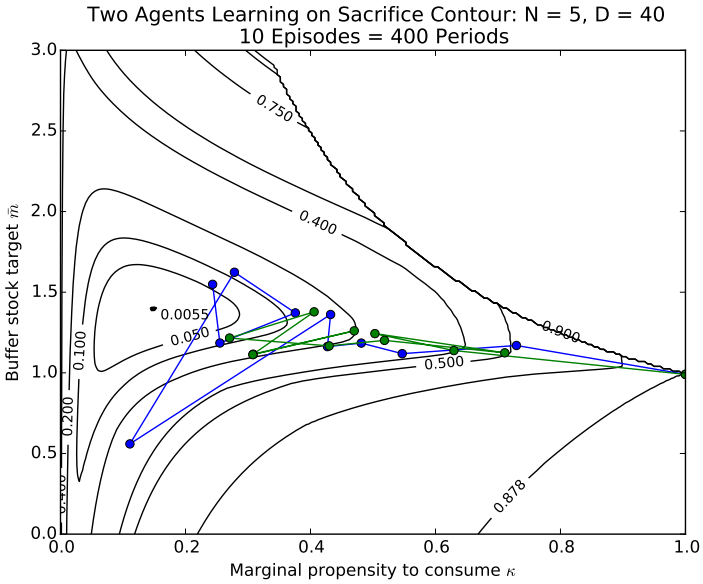


Figure 2.62

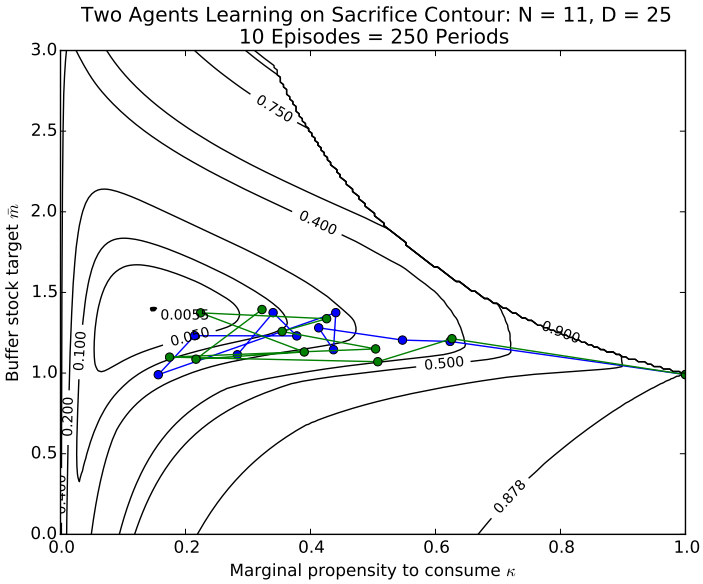


Figure 2.63

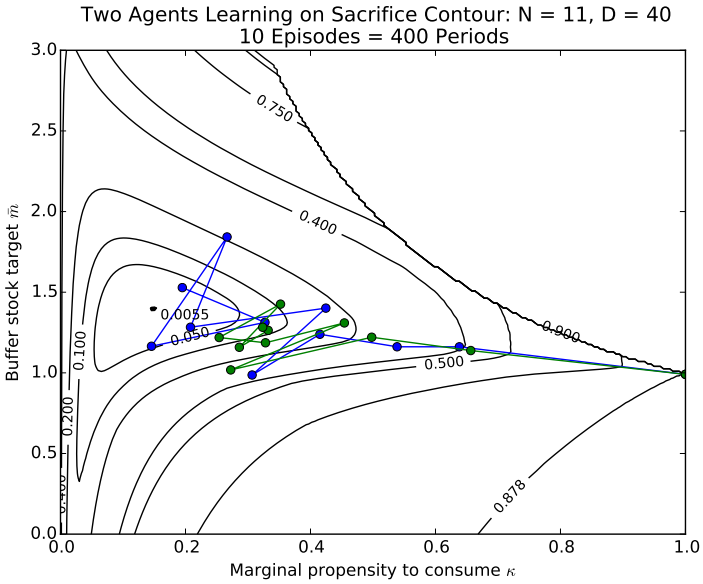


Figure 2.64

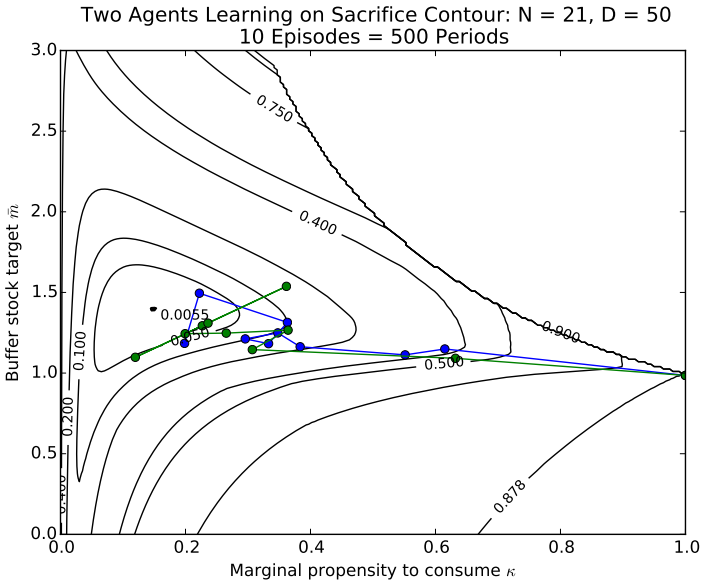


Figure 2.65

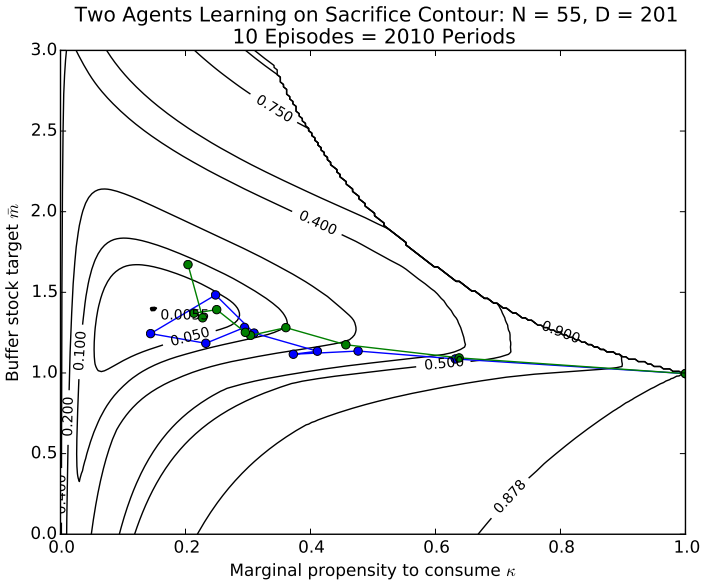

Two Agents Learning on Sacrifice Contour: N = 55, D = 201
10 Episodes = 2010 Periods
Buffer stock target $\check{m}$
Marginal propensity to consume κ
0.750
0.400
0.200
0.100
0.050
0.005
0.999
0.500
0.878
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.2
0.4
0.6
0.8

Figure 2.66

Chapter 3: Social Learning from Experience

3.1 Introduction

Buffer-stock savings is an intuitive and empirically grounded result to emerge from the literature on consumption under uncertainty (Carroll et al. (1992), Carroll (2012c)). Even though buffer-stock savings behavior may be both intuitive and observed, it is difficult to imagine a household coming to this solution via the theory and computation employed by researchers. Allen and Carroll (2001) note this explicitly, and as an alternative, propose that households may learn near-optimal consumption behavior by trial and error. Essentially, they propose that households run a Monte Carlo experiment using their own lifetime experience to estimate the value function associated with a simplified linear consumption rule. They implement this model, and their results are both exciting and discouraging. On the positive side, they show that a simple linear consumption rule can closely approximate the true consumption function. Furthermore, given enough time, consumers using trial and error learning over a set of linear rules can consistently find a near-optimal rule. Their negative result is that "enough time" is anywhere from 200,000 to 4 million years in their model's parameterization. As the authors note, even with a highly efficient search algorithm, the time required to consistently find a near-optimal rule is well outside a human lifetime. The trouble is that the Monte Carlo style estimation relies on random variables which are temporal in nature – a single draw requires agents to literally live through a number of periods.

This paper presents preliminary results for two extensions to the framework of Allen and Carroll (2001) which seek to address the negative result. First, agents are placed on a network and allowed to share their experiences with one another. This greatly reduces the amount of time required for a population of agents to find a near-optimal rule. With

full information sharing among agents, I find it possible to push the number of periods necessary to find a near-optimal consumption rule arbitrarily close to the lowest bound on finding such rules, well within the lifetime of an agent.

The second extension introduces a modified learning rule, such that agents share (and employ) a measure of welfare which is relative to welfare experienced under a "baseline" consumption rule. A natural candidate for the baseline rule is the "consume everything" rule. This extension intends to eventually allow agents with different initial endowments to share information, which is not possible under the original formulation. This is one attempt to make the solution method usable in a larger simulation setting: if endowments can vary, agents can share information freely both across endowments in a single period, and potentially "across generations." Initial results indicate that this modified learning rule performs as well as – and often better than – the original rule in original homogeneous initial endowment setting. In a heterogeneous initial endowment setting, tentative results suggest that the modified rule performs comparably to the original rule in the original setting.

An extensive literature spanning nearly a century exists for the modern formulation of the consumption under uncertainty model. Although Keynes (1936) famously discussed the consumption function in terms of the marginal propensity to consume, it was Friedman (1957) who did much to develop and popularize the idea of the consumption function in its current form, along with the idea that agents acted dynamically and intelligently when making consumption decisions. Due to many technical difficulties, for many years models of the consumption process were bound away from dealing with the issue of income uncertainty – see Hall (1978) for a particularly famous example of this, and Deaton (1992) for an overview of the difficulties this era of models encountered. It wasn't until Zeldes (1989) that the first work on consumption under income uncertainty began to develop. Since then, there has been an explosion of research in this area; Carroll (2001b) offers an excellent overview of this history.

With respect to the particular model extended here, two recent papers have addressed the question of household learning about consumption. Howitt and Ozak (2009) implement

an adaptive consumption rule based on the Euler equation, which does not require extensive memory on the part of the agent. Yildizoglu et al. (2012) explicitly seek to address the same negative results this paper addresses, by expanding agent cognition and learning. Their methods included learning through imitation and learning which employs genetic algorithms and artificial neural networks to form outlooks. My approach differs from both by simply extending the original Allen and Carroll (2001) framework to include social learning without introducing much new cognitive architecture.

The rest of the paper is organized as follows: Section 2 reviews the original Allen and Carroll (2001) model and explains why agent-based modeling (ABM) is the best approach here. Sections 3 and 4 discusses the first and second extensions to the model, respectively, and algorithmic details of their implementation. Section 5 explores the results, and Section 6 concludes with a discussion of work in progress and directions for future research.

3.2 Individual Learning

3.2.1 The Model

Allen and Carroll (2001) construct a simple model of buffer stock savings, as in Carroll et al. (1992). The consumer[1] solves,

$$\underset{\{C_t\}_0^\infty}{max}\ U = \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(C_t) \right] \tag{3.1}$$

[1]The terms"consumer" and"agent" are used interchangeably to refer to the agents in this model.

$$
\begin{aligned}
& s.t. \\
X_{t+1} &= R\left[X_t - C_t\right] + Y_{t+1} \\
C_t &\leq X_t \\
X_0 &= S_0 + Y_0 \\
S_0 & \quad given
\end{aligned}
$$

where S_0 is initial savings, X_t is"cash-on-hand" available to a consumer at time t, C_t is consumption in period t, and R is the interest rate. Here, $R = 1$ for simplicity. Y_t is a simple random income process, (3.2), chosen to roughly match the properties of income explored in Carroll et al. (1992):

$$
Y_t = \begin{cases} 0.7 & with\ prob\ 0.2 \\ 1.0 & with\ prob\ 0.6 \\ 1.3 & with\ prob\ 0.2. \end{cases} \tag{3.2}
$$

The instantaneous utility function will be the standard Constant Relative Risk Aversion (CRRA) form,

$$
u(C) = \frac{C^{(1-\alpha)}}{(1-\alpha)}
$$

with$\alpha = 3$ in this model.

The solution to the problem is the sequence of consumption choices $\{C_t\}_0^\infty$, which may be represented as the optimal consumption function is denoted $C^*(X)$.[2]General properties of the function C^* are known. As discussed in Allen and Carroll (2001), when consumers

[2]I will denote the rule $C^*(X)$ as C^* for brevity.

are impatient enough,[3] C^* can be stated in the form:

$$C^*(X_t) = 1 + f(X_t - \bar{X}^*). \tag{3.3}$$

where $\bar{X}^*$ is a target buffer stock of liquid wealth. When an agent experiences low income, he/she will consume some of the buffer; when income returns to normal, the consumer will attempt to save back to the target level $\bar{X}^*$. The consumption function (3.3), then, is stated as the average expected income (here, 1) plus a function of the**difference** between current cash-on-hand, X_t, and the target buffer stock level, $\bar{X}^*$. Thus if the agent's current cash-on-hand is below the target savings level, f tells the agent how much to consume (and thus how much to save) to move back towards the target savings level.

3.2.2 Approximating a Solution with Experience-based Learning

The function f, and thusC^*, is non-linear and very difficult to find. As Allen and Carroll (2001) note,

> "Despite its heuristic simplicity, the exact mathematical specification of optimal behavior is given by a thoroughly nonlinear consumption rule [C^*] for which there is no analytical formula. While certain analytical characteristics of the rule can be proven, it is hard to see how a consumer without a supercomputer and a Ph.D. could be expected to determine the exact shape of the nonlinear and non-analytical decision rule."

Anecdotal evidence, however, suggests that many people do behave stylistically similarly to the optimal consumption rule – they save some fixed level of income for"rainy days," consume it in bad times, and build it back up in better times. Allen and Carroll (2001) ask a very natural question: if C^* is so difficult to find, how could anyone expect the average household to ever discover it? Certainly the fact that it took casts a dim light on the

[3]The condition in this version of the model is $R\beta < 1$. Intuitively, this is a statement is that consumers are impatient; see Carroll (2012c) for a thorough discussion of this problem.

prospects of the average household ever finding such a rule. Allen and Carroll (2001) offer a novel hypothesis in the literature: if a simple linear version of the rule could be created, perhaps households could get close to the optimal rule by trial and error. To get a simple linear version of (3.3) to use in such a process, Allen and Carroll (2001) take a first-order Taylor approximation around the target buffer-stock wealth, $\bar{X}$, which yielded a function of the form

$$C^{\theta}(X_t) = 1 + \kappa[X_t - \bar{X}], \tag{3.4}$$

and implementing the liquidity constraint,

$$C^{\theta}(X_t) = \begin{cases} 1 + \kappa(X - \bar{X}) & if\ 1 + \kappa(X - \bar{X}) \leq X \\ X & if\ 1 + \kappa(X - \bar{X}) > X \end{cases}. \tag{3.5}$$

where now f has been replaced with a simple linear function κx. For convenience, let $\theta := (\kappa, \bar{X})$ be the pair of parameters which define this rule; the superscript θ indicates thatC^{θ}[4] uses rule θ. I can find the"optimal approximate" rule $\theta^* = (\kappa^*,\ \bar{X}^*)$ by taking κ^* from the linearizion process applied to (3.3); the optimal value for $\bar{X}$ is already know from the optimal solution (3.3). As noted in Allen and Carroll (2001), this optimal rule (to three decimal places) is

$$\theta^* = (\kappa^*,\ \bar{X}) = (0.233, 1.243).$$

Denote the approximate consumption rule that uses these optimal values $C^{\theta *}$.

Both C^{θ} and C^* are possible solutions to the consumption problem (3.1). To discuss consumer welfare, construct the value function associated with each by plugging them into

[4]Because the linear consumption rule C^{θ} is associated with one and only with one parameter-pair $(\kappa, \bar{X})$, I will refer to the linear consumption rule that uses the parameters $\theta = (\kappa, \bar{X})$ interchangeably as both "C^{θ}" and the shorthand, "θ."

(3.1):

$$V^{\theta}(\,X_0\,) \;=\; \mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^t u(C^{\theta}(X_t))\right] \tag{3.6}$$

$$V^{*}(\,X_0\,) \;=\; \mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^t u(C^{*}(X_t))\right], \tag{3.7}$$

with constraints omitted for brevity. By the optimality of C^* for the problem (3.1), I know that (3.7) must be greater than all possible (3.6) values:

$$V^{*}(X_0) \geq V^{\theta}(X_0) \;\; for\; each\; X_0\; and\; \forall\theta.$$

The question is then which value of V^{θ} is closest to the optimalV^*. One may be tempted to examine simple Euclidean distance to choose the closest rule – that is, for each θ, determine

$$\delta^{\theta}(X_0) = |V^{*}(X_0) - V^{\theta}(X_0)|$$

and then simply choose the θ for which δ^{θ} is smallest. This approach, however, is dissatisfying from a theoretical perspective. Technically, the left-hand sides of (3.6) and (3.7) are in terms of"utility," which is actually a reflection of**ordinal** preferences, and therefore the cardinality of (3.6) and (3.7) have no natural value. The cardinality is relevant insofar as it implies an ordinality, but the distances $\delta^{\theta}(X_0)$ have no meaningful value beyond that.

The way around this conundrum is to ask the question"which is better?" in terms of something that is naturally a cardinal value. The approach Allen and Carroll (2001) take is to ask the following:"Assume a consumer is using the optimal consumption rule. How much cash-on-hand would he/she be willing to pay to**not** switch from using the optimal rule C^* to some non-optimal rule C^{θ}?"

Call this value the"sacrifice value," denoted ϵ and defined as,

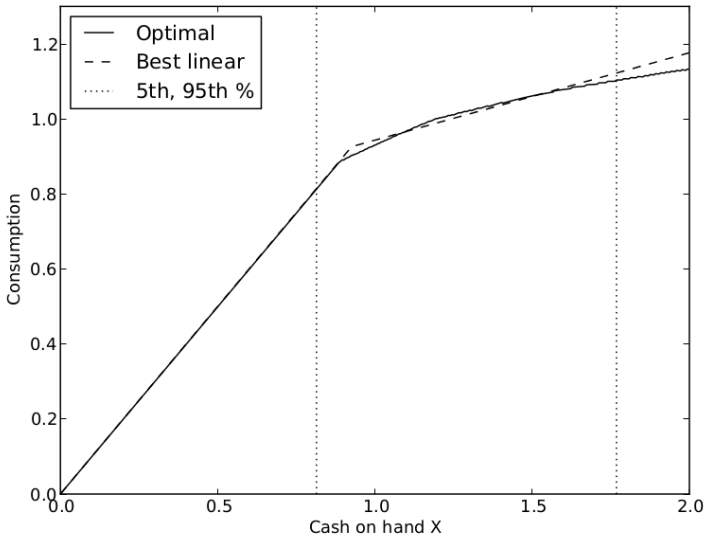
$$\begin{aligned} V^*(X_0 - \epsilon) &= V^\theta(X_0) \\ \Leftrightarrow \epsilon &= X_0 - V^{*(-1)}(V^\theta(X_0)) \\ \Leftrightarrow \epsilon^\theta(X_0) &= X_0 - V^{*(-1)}(V^\theta(X_0)), \end{aligned}$$

where ϵ^θ indicates that this sacrifice value is for rule θ. Note that $V^{*(-1)}$ is the inverse of V^*. Because the sacrifice value clearly depends on the level of wealth that a consumer has at the moment of the decision ,X_0, Allen and Carroll (2001) opt for one additional step. They construct the**average** of the sacrifice value for each rule θ, which they call $\bar{\epsilon}^\theta$:

$$\bar{\epsilon}^\theta = \mathbb{E}[\epsilon^\theta(X_0)].$$

To find this expectation, Allen and Carroll (2001) need a distribution over the X-values. They use the ergodic distribution of cash-on-hand values constructed via simulation as described in Carroll (2001a). They construct a large number of agents with utility and a budget as in (3.1) and an income as in (3.2), and hand them the optimal consumption rule C^*. Then they simply run all these consumers forward through time; after a very short time, the distribution settles down to the ergodic distribution of cash-on-hand wealth.

This brings us to Allen and Carroll (2001)'s first positive results. With $\bar{\epsilon}^\theta$ in hand, a very natural first question is,"what is the sacrifice value of the rule θ^*? Surprisingly enough, they find the answer to be $\bar{\epsilon}^{\theta*} = 0.003$. This is a very small value. Recall that $\bar{\epsilon}^\theta$ is in terms of expected cash-on-hand. Recall also the calibration of annual income in equation (3.2) implies that average annual income is normalized to 1. Thus a sacrifice value of 0.003 corresponds to 0.3% of average annual income for an agent in this model – less than one half of one percent. If, for example, average annual income was instead \$50,000, $\bar{\epsilon}^*$ would be \$150. To state this differently, if it cost a consumer more than a one-time payment of \$150



Author's replication of Allen and Carroll (2001) Figure 2.

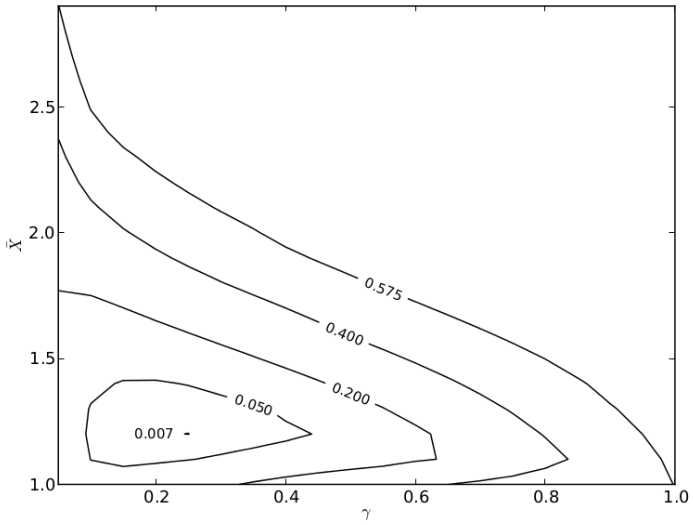
Figure 3.1: The Exact Consumption Rule (solid) and the Best Approximation (dashed)

to find the"thoroughly nonlinear ... and non-analytical decision rule "C^*, the consumer would be better off using the simple linear consumption rule $C^{\theta*}$ for the rest of their lives.

Figure (3.1) reveals why $\bar{\epsilon}^{\theta*}$ is so low: between the 5^{th} and 95^{th} percentiles of the ergodic cash-on-hand distribution, denoted by the vertical dotted lines, the best linear rule is incredibly close to the exact nonlinear rule. Above the95^{th} percentile the approximation grows much worse, but the majority of the time consumers will not encounter this region.[5]

Of course, the optimal approximate ruleθ^* is only one of many possible rules. To get an idea of what the sacrifice value looks like over a wider range of rules, Figure (3.2) displays a contour map of$\bar{\epsilon}^{\theta}$ over a grid of $(\kappa, \bar{X})$ values. The grid is discussed further below. The grid point closest in utility terms to the exact rule is $(0.25, 1.2)$. Call this rule$\theta^{grid} := (0.25, 1.2)$. The sacrifice value forθ^{grid} is 0.007, slightly more than half a percent of average annual income. That is, for a \$50,000 annual income, this would be \$350. Figure (3.2) will provide

[5]The 45° line is due to the fact that over a certain region of cash-on-hand, the optimal consumption choice is actually higher than the agent's income, and thus the agent consumes everything. This is the"credit constrained" region.



Author's replication of Allen and Carroll (2001) Figure 1.

Figure 3.2: Sacrifice Values For Approximate Linear Consumption Rules

important insight into results below, so it is worthwhile to spend a moment exploring it.

Note that the highest-valued contour line, at 0.575, corresponds to the rule$(1, 1)$. Plugging this into (3.4), we get

$$\begin{aligned} C^{(1,1)}(X_t) &= 1 + (1)[X_t - 1] \\ &= X_t. \end{aligned}$$

That is, I get the"consume everything" rule. This rule, in which agents save nothing and always consume their full income each period, has a sacrifice value of 0.575 – for an average income of \$50,000, about \$29,000 dollars. This is significantly higher than the lowest sacrifice rule, whether I consider θ^* or θ^{grid}. The other contour lines are at intermediate values – 5%, 20%, and 40% of average annual income. For an average income of \$50,000, these would correspond to one-time payments of \$2,500, \$10,000, and \$20,000, respectively.

Finding the Approximate Rules

To model trial-and-error learning, Allen and Carroll (2001) propose a statistically consistent estimator of (3.6) by defining$\bar{W}^\theta$ as follows. Let

$$W_i^\theta(\,X_0\,) \quad = \quad \sum_{t=0}^{N} \beta^t u(C^\theta(X_t)), \text{ and} \tag{3.8}$$

$$\bar{W}^\theta \quad = \frac{1}{M} \quad \sum_{i=1}^{M} W_i^\theta(X_0). \tag{3.9}$$

Let us unpack this statement a little. EachX_t is produced by a different realization of a random income process, starting from the sameS_0.[6] As N in (3.8) goes to∞, (3.8) clearly approaches a single-X_t-realization of the summation inside the expectation in equation (3.6). What I want, of course, is the full expectation in equation (3.6). Now I employ equation (3.9). This is the average over M independent"runs" of (3.8), so as $M \to \infty$,$\bar{W}$ will yield a consistent estimator of (3.6).

Thus as$N$ and$M$ jointly go to infinity,$\bar{W}$ yields an accurate estimate the value of a particular consumption rule for a consumer. This can now easily be translated into agents' trial-and-error exploration of consumption rules. Assume that the consumer has some number of different consumption rules to try and compare, say, $\{\theta_1,\ \theta_2,\ \theta_3, ..., \theta_K\}$. Given some rule$\theta$, the household can try it for$N$ periods,M times, and get an estimate of (3.6) for that rule. Repeat this for each rule; then at the end of this procedure, simply compare the values attained by each rule. Pick the highest valued rule and go with that. Since the household clearly cannot let$N$ and$M$ go to infinity, the household will have some noisy estimate of the true value ofV^θ – itself an approximation ofV^* – for some finite values of$N$ and$M$. The question, then, is"what$N$ and$M$ will give consumers a reasonable chance of finding the optimal consumption rule?"

[6] Which produced anX_0 value, as described for the problem (3.1).

To answer this question, Allen and Carroll (2001) create a discrete grid for the consumption rules and instantiate 100 artificial consumers, who explore the space individually with a grid search (discussed further below) for a given(N, M) pair. Specifically, let$\kappa \in (0,\ 1]$ with steps of 0.05, and$\bar{X} \in [1,\ 3)$ by steps of 0.1, for a total of 20x20 = 400 possible rules. Formally, the rule spaceΘ which Allen and Carroll (2001) employ is

$$\Theta := \left\{(\kappa,\ \bar{X})\ s.t.\ \kappa = \{0.05, \ldots, 1.0\} \text{ in 0.05 steps};\ \bar{X} = \{1.0, \ldots, 2.9\} \text{ in 0.1 steps}\right\}. \quad (3.10)$$

Allen and Carroll (2001) sweep over a parameter space of$N$ and$M$, repeating the above process for each(N, M) pair in the space:

$$\mathbb{L} := \{(N, M)\ s.t.\ N \in \{10, 20, 50\}\ and\ M \in \{1, 10, 50, 200\}\}.$$

Lastly, because the process of (3.8) and (3.9) requires the same value of initial cash-on-hand forX_t, Allen and Carroll (2001) restrict their consumers to repeated searches from the same initial savings value,S_0 (recalling the budget constraint in equation (3.1) above); this ensures that (3.9) is a consistent estimator of (3.6).[7] To see whether differentS_0 values influence how well a consumer can find an optimal rule, Allen and Carroll (2001) include S_0 as a parameter over which to sweep.

3.2.3 Original Model Results

The results of Allen and Carroll (2001)'s parameter sweeps across N and M are displayed in Table3.1, an excerpt of the original results.[8] "Mean Sacrifice" is $\bar{\epsilon}^{\theta}$. "Success Rate" measures the fraction of the 100 consumers who have chosen a rule with a sacrifice value ≤ 0.05 by the end of the simulation – that is, the consumers who fall within the 0.05 contour in Figure 3.2. "Total periods" is the number of periods it takes a consumer to obtain the solution.

[7] They useS_0 instead ofX_0 to make some technical computations easier.

[8] The full results include tables for $S_0 = 0$ and $S_0 = 2$ as well. These results are largely in line with those shown, and are excluded in this paper for the sake of space and clarity.

Table 3.1: Allen and Carroll's Individual Search Results

S0 = 1					
		M = 1	M = 10	M = 50	M = 200
N = 10	Mean sacrifice:	0.269	0.122	0.100	0.102
	Success rate:	0.09	0.23	0.29	0.24
	Total Periods:	4000	40000	200000	800000
N = 20	Mean sacrifice:	0.226	0.079	0.053	0.047
	Success rate:	0.18	0.45	0.62	0.68
	Total Periods:	8000	80000	400000	1600000
N = 50	Mean sacrifice:	0.187	0.058	0.036	0.024
	Success rate:	0.26	0.58	0.76	0.91
	Total Periods:	20000	200000	1000000	4000000

Source: Allen and Carroll (2001)

It is immediately clear that for consumers to find a near-optimal linear rule**reliably** – greater than 67% of the time, for example – they must spend hundreds of thousands of years exploring the parameter space. The lowest (N, M) pair which achieves this is $(50, 50)$, which takes 1,000,000 years for each agent to undertake individually. This result turns more negative when I recall that these outcomes apply for only a single value of S_0 – if consumers wanted to look for a best rule across multiple values of S_0, they need to roughly triple their search time. Clearly, part of this time is the inefficient grid search – the fact that each consumer must try out each of the 400 rules immediately implies that the time spent for a given (N, M) combination is $N * M * 400$. Note, however, that a more efficient search will not solve the problem. As Allen and Carroll (2001) state,

> "...even if the search could be reduced so that only, say, 4 different rules needed to be evaluated, it would still be necessary to use values of (M, N) large enough to distinguish good rules from bad. Given that the minimum (M, N) combination that appears capable of producing the necessary accuracy is (50,50), even [a] highly efficient hill climbing routine could not reduce the number of periods required to less than $10{,}000 = 50 * 50 * 4$."

The key is really what values of (N, M) provide an acceptable**success rate**; which (N, M)

pair allows the consumer to distinguish good rules from bad rules using $\bar{W}^{\theta}$. The top panel of Table3.2 shows that, for a**minimum**success rate greater than 67%, we still need an (N, M) pair of at least $(50, 20)$.[9] Even if agents only need to explore 4 rules, this still implies that an agent would need to take 10,000 years to find an adequate rule. The story is broadly the same for $S_0 = 0$ and $S_0 = 2$ (not shown).

Thus Allen and Carroll (2001) arrive at two strong positive results and one strong negative result:

1. A linear approximation to the optimal consumption function can get negligibly close to the optimal rule in utility terms;

2. given enough time, consumers can **reliably** find a near-optimal rule with simple trial-and-error learning, and

3. unfortunately, "enough time" is prohibitively high – anywhere from 400,000 to 4 million years.

3.3 The First Extension: Social Learning

It is clear that the trouble faced by agents in this model is akin to that of a programmer completing a large number of independent tasks in serial on a single machine. If there are 50 independent, identical tasks and 50 available processors for parallel computation, the job could be done in $1/50^{th}$ the time. This relies, of course, on the fact that the 50 tasks

[9]The replication of the model shown in the middle panel of Table3.2 disagrees slightly with the original results in the top panel (and Table3.1) regarding which (N, M) pair is needed to achieve a minimum success rate of 67%. The model of Allen and Carroll (2001) required $(50, 50)$, while the replication only required $(20, 50)$. It is clear in their results that $(20, 50)$ is right on the edge of the 67% success rate, and Figure D3 (discussed further in Section 5) makes it clear why these numbers may easily vary: the slope of the"Success Rate" line for $(20, 50)$ is incredibly steep at the cutoff value of 0.05. One can see that even slight differences due to the randomness inherent in the experiment could shift this line around. For the sake of consistency, from here forward I will refer to the replication results of $(N, M) = (20, 50)$ as the parameter pair at which agents can expect to achieve a 67% success rate.

are independent of one another. Recall the estimation problem:

$$W_i^\theta(\,X_0\,) \;=\; \sum_{t=0}^{N} \beta^t u(C^\theta(X_t)), \;\text{ and}$$

$$\bar{W}^\theta \;=\; \frac{1}{M}\sum_{i=1}^{M} W_i^\theta(X_0).$$

Estimating W_i^θ is an"embarrassingly parallel" problem if each experience of W_i^θ is independent. Fortunately, independence of the W_i^θ experience is a requirement of the original model, for $\bar{W}^\theta$ to be a consistent estimator. Furthermore, the"lifetime experience" of the 100 agents occurs in parallel (in"agent time"). The problem is perfectly suited to be parallelized among the agents.

To do so, I need to specify a method of sharing information between agents. I will need to answer the question of which information is shared, whether any is lost in the process, and with whom agents will share their information – that is, the network of information sharing. As a first pass, I consider and implement the simplest possible case: agents share information perfectly with all other agents.While this may seem unrealistic, it allows us to create an"upper bound" on what may be achieved in this model with information sharing.

The information the agents share is the value W_i^θ, which they simply hand off to their neighbors (i.e. all other agents) as soon as they finish estimating it. As it would be a waste of time to explore a rule that has already been explored M times, I need to implement a mechanism for coordination among agents. The next section explains the implementation of the information-sharing system, which is written to be flexible and allow immediate computational exploration of alternate behaviors.

The difference between this and the original model is that now agents are**social agents**, as well as being intelligent agents. Their interactions with one another materially affect the outcomes of the experiment in very positive ways, as will be shown.

3.3.1 Implementation of Full Information Sharing

The general mechanism I use to achieve coordination among agents, on any network topology, is a"bulletin board" and a list of neighbors. The bulletin board is essentially a hash table with an entry for each rule θ; each rule's entry contains an array of W_i^θ values. Each agent owns a bulletin board, which is initially empty, a list of neighbors, and an income process. When a given agent,k, finishes a single N-length estimation of a value W_i^θ, he stores that value under the appropriate rule in his own bulletin board, then pushes the value to all of his neighbors, who store it in their bulletin board. In a not-fully-connected network of agents, there is a question of whether the neighboring agents then pass that information along. In a fully connected network, this is not a concern.

When selecting the next rule, agent k simply chooses a random rule off his board which does not yet have M values in its array. Thus coordination is automatically obtained.If agents are exploring the rule-space in a synchronized fashion, coordination could be costlessly obtained by simply choosing the next rule after all agents update their neighbors. In an asynchronous world, there is a trade-off between waiting to hear from a neighbor (and perhaps saving oneself N periods of rule exploration) and simply starting to explore a random uncompleted rule. This trade off, however, is beyond the scope of this paper to consider.The model terminates when all agents have obtained the number of W_i^θ experiences necessary to estimate all $\bar{W}^\theta$ values.

The above applies in the general case, where the network of agents is neither degenerate ("individual learning") nor fully connected. In the special cases of individual learning and fully connected social learning, agents do not need to use their list of neighbors. This is obvious in the individual learning case; in the fully connected case, it is simpler to assign each agent an empty neighbor list and a reference to a global bulletin board. Coordination and full information sharing then occurs automatically as the algorithm executes.

This model is constructed in Python, employing the NumPy and SciPy scientific libraries when necessary. The object-oriented framework provided by Python is well suited for

constructing the bulletin boards each agent uses, and each agent encapsulates their own information and behavior, allowing alternate behaviors to to be quickly introduced. This is aspect of the code is exploited in the second extension, below. Lastly, the language allows the model to be easily run on"cloud" parallel computing platforms, greatly decreasing final execution time.

3.3.2 Algorithm for Full Information Sharing

To run the model, 100 Consumer agents, a Bulletin Board object, and 100 Income Process objects are created. Each agent is handed an income process and a bulletin board. Each period, the following process occurs:

1. Agents are selected in a random order to try out a rule.
2. Upon selection, each agent chooses a random rule and checks their bulletin board to see if it has already been tried$M$ times.
   (a) If so, another rule is selected until one is found which hasn't been tried M times. If such a rule can't be found, the simulation terminates for that agent.
   (b) If an undepleted rule can be found, the consumer tries the rule for N periods and posts the resulting W_i^θ value to his bulletin board.
3. Repeat the process until all agents terminate their efforts.

3.3.3 Results for Full Information Sharing

In the full information sharing setup, once every rule has been tried M times, the entire population of agents will choose the same highest-value rule, as they all share the same bulletin board. To learn how well this"full connection" social learning process can be expected to work, and in order to compare these results to those in Table3.1, I run the"full connection" experiment 100 times for each M, N pair. Thus I can make a statement about what fraction of the time a population using this process can be expected to fall

within $\bar{\epsilon}^{\theta} \leq 0.05$ of the optimal rule, analogous to the information expressed in Table 3.1. In fact, besides changing the number of periods it takes an individual consumer to to determine an optimal rule, this should be mathematically identical (accounting for some random variation) to the"individual learning" process outlined in the original study. This is because the"full information sharing" version of the model described above nests the original Allen and Carroll (2001) model as a special case. If we instantiate the social learning model with only one agent (instead of a population of agents) and run the model to completion 100 times, the single agent will clearly be forced to explore the entire parameter space alone in each of 100 runs. This replicates the structure of the original Allen and Carroll (2001) model entirely.

The results of the full information sharing extension are displayed in the middle panel of Table3.2 and as the green line with triangles in Figure 3.2the y-axis plots the success rate and the x-axis plots the$M$-values from the table. For ease of reference, the original results from Allen and Carroll (2001) are displayed in the top panel of Table 3.2 and as the black line in Figure 3.2. As is expected, the success rate for Full Connection Social Learning model is very close to the success rate of the original Allen and Carroll (2001) model – this is illustrated in Figure 3.2. The main difference arises in the number of periods necessary to obtain these near-identical success rates. These can be observed in the"Total Periods" row in the top and middle panels of Table 3.2. Consider the pair $(N, M) = (20, 50)$. The original model took 400,000 periods of agent search time to distinguish good rules from bad rules at least 67% of the time. For the full connection model, the equivalent time was 10,000 periods – reduced by a factor of 100. This is entirely due to the fact that 100 agents worked together, in parallel, to explore the parameter space. Note that, with 100 agents and 400 rules, if the agents split up the rules evenly, each would have on average only 4 rules to explore – thus achieving a similar result to the Allen and Carroll (2001) quote above, only using social learning instead of a"highly efficient hill climbing routine." In fact, if more agents were added to the full-connection model, this number can be reduced further. If one agent were added to the model for each possible run-trial – $(N, M) = (20, 50) \implies 50 * 400 = 20,000$

agents – then the time required to search the entire space would only be bounded by the time it took to complete a single W_i^θ trial: 20 periods.

This result hinges on the fact that all agents are sharing information fully with all other agents – essentially, that all agents are in a fully connected social network. In practice, of course, it is hard to imagine that 20,000 agents all know each of the other 19,999 agents in a population. The discussions and case studies cited in texts such as Tsvetovat (2011) and Newman (2010) clearly indicate that common human social networks are nowhere near fully connected graphs. The purpose of this full connection model, however, should be viewed as constructing a"ceiling" on how well agents can do if they can share information amongst themselves. The original Allen and Carroll (2001) model sets something of a"floor" on how poorly agents can do if they share no information whatsoever. If it was the case that the full connection model was unable to bring agent search time within some reasonable limit, then clearly there would be nothing left to explore of any of the myriad of possible network topologies or information transmission mechanisms. As it stands, however, the full information sharing experiment strongly indicates that social learning can greatly improve individual agent learning. Some combination of network structures and information sharing mechanisms may very well bring agent search efficiency very close to the full connection model. For example, a few highly connected agents with a high transmission rate may produce very similar results; such a network may very well represent a world in which"financial experts" specialize in gathering, analyzing, and disseminating such information.

Before I discuss further possible research along network lines, however, I turn my attention to my second extension to Allen and Carroll (2001)'s original framework: considering"relative" rather than"absolute" happiness.

3.4 The Second Extension: A Relative-Value Estimator

Returning to the general model setup, recall that agents are estimating an approximation to the value function V^θ in equation (3.6). This value function is a measurement of the

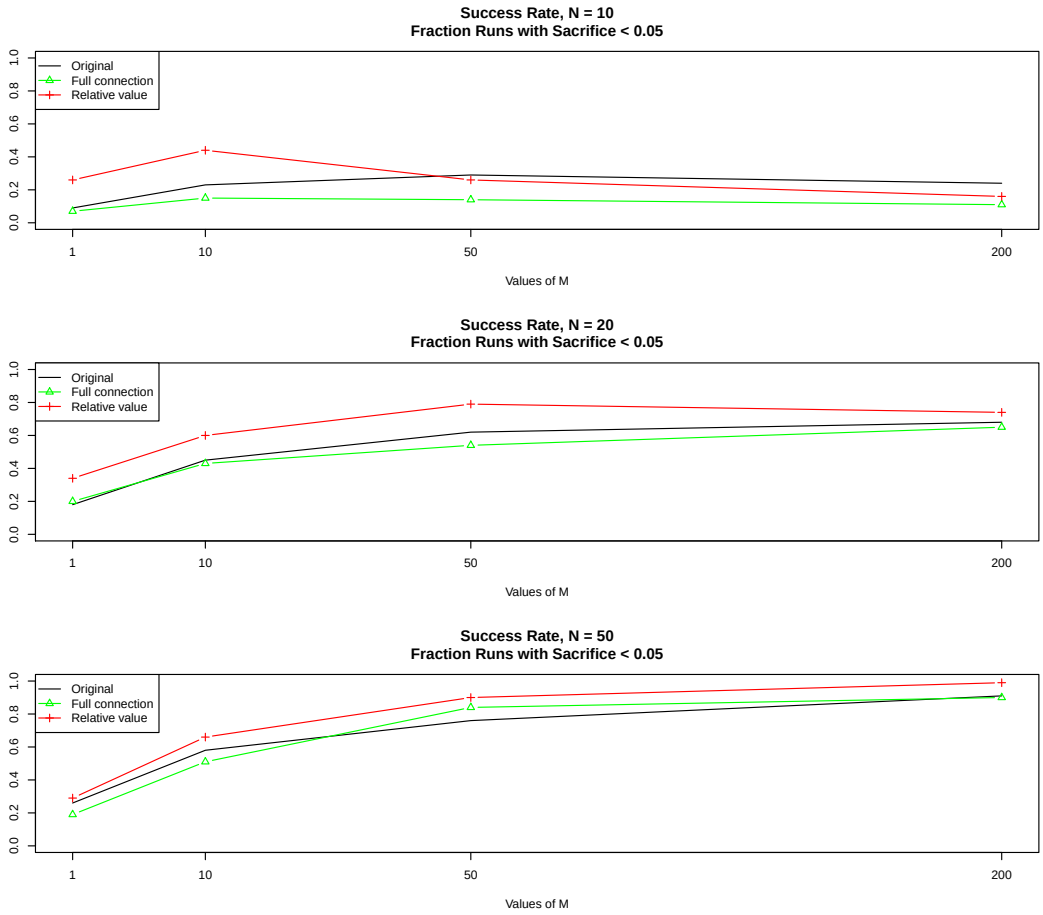


Figure 3.3: Allen and Carroll Results vs Full Connection and Relative-Value Learning

agent's"happiness" with units of this"happiness" are in terms of expected utility. There is, however, another way to think about"how well" a rule works. Instead of asking"how happy does this rule make me," agents might instead ask,"how happy does this rule make me**above and beyond** how happy (or unhappy) I would have been using some other rule?" This entails choosing some baseline rule against which agents compare their current experience. I will refer to this as a"relative-value" measure of well being.

3.4.1 Implementation of the Relative-Value Estimator

A natural baseline rule is the"consume everything" rule. Thus instead of estimating the "absolute" value function,

$$V^{\theta}(\,X_0\,) = \mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^t u(C^{\theta}(X_t))\right]$$

agents instead employ a relative-value estimator, which simply subtracts off the value an agent would have obtained had he/she employed the"consumer everything" baseline rule:

$$\tilde{V}^{\theta}(\,X\,) = \mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^t u(C^{\theta}(X_t)) - \sum_{t=0}^{\infty}\beta^t u(C^{Base}(X_t))\right].$$

As before, the agent estimates this function by experiencing income streams and recording the results. Each consumer will now estimate the following value:

$$\begin{aligned}\tilde{W}_i^{\theta}(\,X_0\,) &= \sum_{t=0}^{N}\beta^t u(C^{\theta}(X_t)) - \sum_{t=0}^{N}\beta^t u(C^{Base}(X_t)), &(3.11)\\ &= \sum_{t=0}^{N}\beta^t\left[u(C^{\theta}(X_t)) - u(C^{Base}(X_t))\right], \text{ and}\\ \bar{W}^{\theta} &= \frac{1}{M}\sum_{i=1}^{M}\tilde{W}_i^{\theta}(X_0). &(3.12)\end{aligned}$$

with $\bar{W}^{\theta}$ calculated as before in equation (3.9). $C^{Base}(X_t)$ is the baseline"consume everything" consumption rule. Thus $\tilde{W}_i^{\theta}$ is the relative-value estimator, and the original W_i^{θ} is the absolute-value estimator.

An additional intended benefit of this relative-value consumption rule is that it may allow for heterogeneous S_0 values at the beginning of each estimation o f$\tilde{W}_i^\theta$. Recall from our discussion of equation (3.8) that each estimation run restarts with the same initial S_0, as in the original model. As it stands, for an agent exploring some rule θ in an $\{N, M, S_0\}$=$\{50, 10, 1\}$ experiment, I must think of the agent as"re-setting" his/her savings to the same S_0 value every 50 periods. If I want agents to carry savings from one N-period stream to the next, as I might imagine an agent doing in practice, I must somehow normalize the initial endowment at the beginning of each estimation of a $W_i^\theta(X)$. If I don't normalize for heterogeneous S_0, a high initial value may skew results in favor of whichever rule just so happened to enjoy the"high S_0." Observe that β^t is a larger number when t is small – that is, early on in the trial of $\tilde{W}_i^\theta$, when S_0 matters the most. If an agent just so happens to have a high value of S_0 for a"bad" rule, but a low value for a"good" rule, the good rule may **appear** to be be bad, simply because of the discounting of β^t over the N periods.[10] If the selection of S_0 is random, this may not be a problem at higher values of M, but if S_0 is not random this may bias results. The relative-value estimator, $\tilde{W}_i^\theta$, seeks to address this by subtracting off the"consume everything" utility and thus normalize agent experience due to initial S_0 values.

3.4.2 Algorithm for Relative-Value Information Sharing

As a first step towards using the relative-value estimator for experiments with heterogeneous S_0, this procedure is first implemented in the same homogeneous S_0 setting as the absolute-value estimator with full information sharing. If the results of this implementation are much better (or much worse) in the homogeneous-S_0 setting, this can indicate of how well the estimator can be expected to perform when used in a heterogeneous-S_0 setting. Thus algorithmic implementation of the the relative-value estimator is exactly the same as Section 3.3.2, except that the relative-value measure $\tilde{W}_i^\theta$ is used instead of W_i^θ.

[10]Note that even if S_0 is held constant across all runs, the fact that X_0 is the sum of both S_0 and a random Y_0 means this dynamic may still play out, albeit to a lesser extent.

HERE HERE

3.4.3 Results for the Relative-Value Estimator

The results of the relative-value process are displayed in the third panel in Table 3.2, and in the red line with plus signs in Figure 3.3. The the"Total Periods" values in Table 3.2 are the same for both the absolute-value and relative-value estimators; this is to be expected, as both of those experiments have been run in the full-connection social learning setting. What is remarkable, however, is that the relative-value estimator has actually made it easier for consumers to distinguish good rules from bad rule. Previously, in the absolute-value experiment (the second panel in Table3.2), consumers needed at least an $(N, M) = (20, 50)$ pair to achieve a minimum 67% success rate. This corresponded to 4,000 periods of search. In the relative-value experiment, however, consumers have improved their rule-finding capabilities by a full order of magnitude: now they can achieve a minimum success rate of 67% at either $(N, M) = (10, 20) \implies$ 800 periods. This is less than half that experienced using the absolute-value rule.

The results are more striking when observed in Figure3.3. Without fail, the relative-value success rate is above the success rates of the original model and the social learning extension. That is, for every possible (N, M) pair, the relative-value estimator allows more experiments to fall within the $\bar{\epsilon}^{\theta} \leq 0.05$ cutoff range. To state it differently, this estimator allows the consumers to consistently distinguish"good" rules from"bad" rules, at every (N, M) value.

3.5 A Closer Look at Model Output

3.5.1 Examining Distributional Output

To understand why the relative-value estimator works so well, I need to dive deeper into the functioning of the model. Fortunately my model is a computational simulation, and thus it may be run many times to generate distributions of output. I do this in Figures D1

and D2, in Appendix D. These figures are representations of two-dimensional histograms: each dot represents a number of agents that settled on a given rule $\theta := (\kappa, \bar{X})$ in the Θ space described in equation (3.10); the area of the dot is proportional to the number of agents at a particular rule on the grid.[11] This is overlaid on the contour plot displayed in Figure3.2. Thus I have a non-parametric way to examine the entire space of results, instead of only a slice of the results, as displayed in Table3.2. In all of the histograms, the optimal approximate rule, $\theta^* = (0.233, 1.243)$, is represented by a red"+," and the optimal point on the grid, $\theta^{grid} = (0.25, 1.2)$, is represented by a black"x."

In Table3.2, the success rate refers to the fraction of agents which fall within the 0.05-contour in Figures D1 and D2. A few stylistic facts emerge as I examine these histograms. In Figure D1, start at the panel corresponding to $(N, M) = (10, 1)$. It is clear that as I move to the right, examining panels $(10, 10)$, $(10, 50)$, and $(10, 200)$, the agents are choosing rules in a tighter groups – the problem is that the rules they are converging on are**just** outside the 0.05-contour. The fact that the agents converge on a location that is not quite the optimal target is stylistically similar to the idea that an estimator can have high bias but low variance. As M increases, the variance drops but the bias does not. If, however, I again start at the upper-left panel $(10, 1)$ in Figure D1 and move down, examining panels $(20, 1)$ and $(50, 1)$, a different story emerges – the cluster of rules the agents arrive at may not tighten as it did along the top row, but it does shift to center around the optimal θ values. This is more apparent when the agents' final choices are less diffuse, for example in the rows corresponding to $M = 50$ or $M = 200$. For the row $M = 50$, it almost appears that the diffusion may increase slightly. The main difference, however, between $(10, 50)$ and $(50, 50)$ is that at $N = 50$, the histogram is centered around the optimal value, instead of a nearby value as at $N = 10$. This makes a significant difference for the success rate. This is

[11]These Figures are arranges to correspond in layout to the tables of output in Table3.2. Although analysis may be run with a high number of agents and model runs, the results shown here are for 100 runs of 100 agents with full information-sharing. Recall that the time spent examining the rule space is only difference between the original Allen and Carroll (2001) model and the version employing full information sharing and the absolute-value estimator. Thus the histograms in Figure D1 represent both the original Allen and Carroll (2001) model as well as the absolute-value estimator version of the model.

stylistically similar to the idea that an estimator can have low variance but high bias. Of course, as both N and M increase, both the bias and the variance decrease. These dynamics are all entirely hidden when I only examine the Fraction Success rows in Table3.2.

Given that I can generate distributions from the model, these ideas of bias and variance in the histograms may be examined more rigorously statistically. This is left for future work, however. For current purposes, this initial stylistic understanding is enough to reveal some important first-order questions. It is clear from the discussion above that there is a trade-off occurring in between"bias" and"variance" in the model results. The choice of the $\bar{\epsilon}^{\theta}$ cutoff of 0.05, which I use to determine the success rate in Table3.2, appears to be significant when I consider the histograms in Figure D1. I might ask whether the choice of cutoff value makes a significant difference in my results. If so, I need to carefully consider how the choice of cutoff value is made. For panel $(20, 1)$ in Figure D1 a cutoff value of 0.05 instead of 0.075 or 0.1 may not make a significant difference. For panel $(20, 10)$ in the same Figure, however, it may make a large difference.

Before continuing, observe that examining Figure D2 provides an initial indication regarding why the relative-value estimator preforms better than the absolute-value estimator. Broadly speaking, for the absolute-value estimator, the histograms tend to"spread" more along the the horizontal κ axis – this is most apparent in the first two rows of Figure D1, for example. For the relative-value rule, however, the histograms have been shifted – they tend to"spread" along the vertical $\bar{X}$ axis. The practical result of this is that the"teardrop" shape of the $\bar{\epsilon}^{\theta} \leq 0.05$-cutoff region captures more of the relative-value histograms earlier than it captures the absolute-value histograms for a given (N, M) pair. The relative-value histograms in Figure D2 cluster more around the $\bar{\epsilon}^{\theta} \leq 0.05$-cutoff region's wide"base," while the absolute-value histograms in Figure D1 cluster around its narrow"top." As with the bias-variance trade-off above, this complicated relation ship may very well be explored more rigorously, either statistically or analytically, but this is left for future work. The stylistic impressions these histograms and contour plots provide are sufficient to motivate the next step in exploring the model. Furthermore, it is not clear that any simple or informative

closed-form analytical expression exists which may describe either the bias-variance trade-off or the relationship between the histograms and the contour plots in Figures D1 and D2.

3.5.2 Examining the $\bar{\epsilon}^{\theta}$ Cutoff Value

A first-order question that arises from considering the histograms in Figures D1 and D2 is whether the choice of cutoff value 0.05 for $\bar{\epsilon}^{\theta}$ is significant for the success rates displayed in Table3.2. The histograms cannot supply any further answers with more precision so the question must be taken to another representation of the data. Figures D3 and D4 in Appendix D display the success rate along the vertical axis as a function of possible cutoff values along the horizontal axis. From Figures D1 and D2, it is clear that very few agents choose rules outside of the cutoff value $\bar{\epsilon}^{\theta} = 0.575$, the value corresponding to the consume-everything rule.[12] Thus the horizontal axes only extends to 0.5.

The dotted gray line at the bottom of the legend in each plot indicates the original cutoff value of 0.05 employed to determine the success rates in Table3.2. The red dash-dotted line above the 0.05 line in each plot illustrates what the success rate would be if the cutoff were instead 0.075, and the dashed line indicates what the success rate would be if the cutoff was 0.1. These correspond to 7.5% and 10% of average annual income. In the earlier example of a \$50,000 average income, these values would be \$3,750 and \$5,000, receptively. The solid horizontal line with stars on it indicates the cutoff value that is associated with a success rate of 0.5 across all (N, M) pairs. Thus in Figure D3, in panel $(10, 1)$, agents would need a cutoff value of 0.31, roughly $\frac{1}{3}$ of average annual income, to achieve a success rate of 67%.

It is clear from Figure D3 that the curvature of the"success rate" line is important. Each increase in the cutoff value"buys" additional"success," and the nonlinearity in the success rate function determines how big this marginal increase will be. As noted above, for panel $(20, 1)$ in Figure D3 a cutoff value of 0.075 or 0.1 instead of 0.05 does not significantly shift

[12]It is apparently possible to do**worse** than simply consuming everything and saving nothing, but agents rarely select one of these worse rules in practice.

the success rate greater than 67%. However, for panel $(20, 10)$ in the same Figure, a cutoff value of 0.075 or 0.1 brings the success rate right to 67%. This does change the results of Table3.2. Looking at Figure D3, it is clear that if I were to consider the cutoff value of 0.075 for the absolute-value estimator consumers would be able to achieve a 67% success rate at $(N, M) = (20, 10)$ instead of $(20, 50)$ – in 800 periods instead of 4,000 periods, if each consumer only had to explore 4 rules (either through a"highly efficient hill-climbing routine," or through fully connected social learning with 100 agents in parallel). This is a significant improvement in the success rate of the algorithm, simply by considering a slightly increased cutoff value.

What I would like to see, of course, is a 67% success rate for a parameter pair such as $(N, M) = (10, 1),\ (20, 1)$, or $(50, 1)$. Such values would imply that consumers could reliably find"good" consumption rules within 1 to 3 generations, if a generation lasted roughly 70 periods. This brings us to the results of the relative-value estimator, displayed in Figure D4. A difference which is immediately clear between the absolute-value estimator and the relative-value estimator in Figures D3 and D4, respectively, is the shape of the"success rate" line for the parameter-pairs $(N, M) = (10, 1),\ (20, 1),$ and $(50, 1)$. In Figure D3, $(10, 1)$ and $(20, 1)$ are almost linear, which is why increasing the cutoff value does not much improve agents' ability to distinguish good rules from bad rules. In Figure D4, however, all three curves in the first column of panels gain a lot of curvature over their their counterparts from the absolute-value estimator. Now increasing the cutoff value has a significant impact on how well the agents distinguish good rules from bad rules. In fact, if I am willing to choose a cutoff of 0.16, I can obtain a success rate of 67% at the lowest possible $(N, M) : (10, 1)$. If I choose a cutoff of 0.12, I can obtain the desired success rate at $(20, 1)$. If an agent only needs to explore 4 rules (either via efficient search or social learning), this translates into agent time of only 40 and 80 years, respectively.

There is a clear trade-off between obtaining an acceptable success rate with less agent effort by raising the cutoff value, and the value lost to the consumers which is directly represented by the subsequently higher cutoff value. An agent who"succeeds" because the

cutoff value has been raised to 0.12 is potentially sacrificing 7% more of an average annual income than an agent who simply "explores longer" and "succeeds" with the original cutoff value of 0.05. There may very well be some optimal cutoff value to choose. The simulation code allows many possible experiments to be conducted with this model; this is one possible question that may be addressed in future research. Regardless of whether or not an optimal cutoff value exists the important takeaway is that the choice of cutoff value matters for the results of the model, and needs to be considered a parameter of the model alongside the others. Figures D3 and D4 are an attempt to sweep over this parameter as well; the reader may examine Figures D3 and D4 and see for himself or herself what the influence of the cutoff value is on model results.

3.6 Conclusions and Future Work

3.6.1 Summary and Conclusions

Allen and Carroll (2001) uncovered both striking positive and striking negative results. Using a simple linear approximation to the optimal consumption rule,they found that agents can get negligibly close (in utility terms) to the highly non-linear optimal solution. In addition, given enough time, agents could consistently find a near-optimal linear rule via simple trial-and-error search. Unfortunately,"enough time" proved to be prohibitively long – as Table3.2 indicates, to reliably find a near-optimal rule – that is, achieve a success rate of at least 67%[13] – agents need to spent 400,000 years searching the parameter space.

This paper addresses the final negative result in two separate steps, each of which greatly reduces the time required to reliably find a near-optimal rule. The first step takes its inspiration from Allen and Carroll (2001) themselves. In their 2001 paper, they state that

> "If it takes an individual agent a million periods, ...a population of a million

[13]Or very near 67%; see the footnote at the end of Section 2.3.

> consumers... should collectively obtain essentially the same amount of information in a single period."

In addition they note that,

> "More intriguing [than efficient search algorithms] is the possibility that consumers come by their behavior by a process of social learning, in which rules of thumb that are successful in utility terms are passed along from one consumer to another, or through other mechanisms such as the advice of personal finance experts or advice in personal finance books. ... Elucidating the circumstances under which a process of social learning can be expected to lead the population to reasonably optimal behavior will be an interesting task for future work."

Here and elsewhere, Allen and Carroll (2001) suggest two possible extensions to their original model: (1) implementing a more efficient search algorithm over the grid space of possible rules (alluded to above), and (2) some form of social learning. Taking inspiration from the second of these suggestions, I first observed that the agent estimation problem is an embarrassingly parallel computation. I then constructed artificial agents in code and provided them with a bulletin board to store the results of rule-trials, and a network of neighbors with which to share their experiences. When an agent tries a rule, he/she stores it in his own board, and also passes it to all his neighbors. As a first-pass exploration, and in order to have results mathematically comparable to the original model, I instantiated 100 consumers and placed them all on a fully connected social network; the model is then run 100 times and results recorded. This extreme network structure nests the original Allen and Carroll (2001) model as a special case (obtained when there is only 1 consumer on the fully connected network) and allows us to explicitly discuss the improvement provided by social information sharing.

The result is that the"Total Periods" in the middle table of Table3.2 are reduced by a factor of 100 from those of the Allen and Carroll (2001) experiment. Where once it took a minimum of 400,000 years to get a success rate greater than (or very near) 67%, in

the full connection experiment it takes only 4,000 periods. The reduction of total periods follows immediately once similar success rates are obtained between the original and the full information sharing procedures. In fact, this result confirms that the"total periods" could by pushed arbitrarily close to the lowest bound for each M, N pair by simply adding more consumers. The lowest bound is simply the N-value, since a rule must be tried at least this many years. This is an encouraging result, and the first step in a multitude of possible research directions. Real networks, of course, do not have anything near a fully connected structure; strategies for addressing this, however, will be discussed further below.

The second step in addressing the Allen and Carroll (2001)'s negative results was to create an estimator of the consumer's value function**relative to a baseline** consumption rule. This estimator takes its inspiration from the idea that an agent is likely to think of his happiness not in some absolute terms, but rather in relation to some baseline course of action. Furthermore, in a technical sense, the relative-to-baseline value function is also an attempt to normalize the estimator in the face of heterogeneous initial wealth conditions. If successful, this would allow agents to be instantiated with heterogeneous wealth levels during the experiment. An encouraging result is that this estimator not only works as well as the original absolute-value estimator, but also improves upon its performance. Agents' ability to distinguish"good" rules from"bad" rules improved when they used the relative-value estimator to explored the parameter space.

In addition, the improvement provided by the relative-value estimator is orthogonal to the improvement provided by the social learning mechanism. As discussed in Section 3, the improvement provided by the social learning mechanism was entirely in the"Total Periods" rows of Table3.2. The"Success Rate" between the top table and middle table of Table3.2 was determined by the same fundamental process, in particular because the experiment that produced the middle table nested the original results as a special case. They only appear different due to random variation. For the relative-value estimator, however, the fundamental estimation process has changed, resulting in a higher success rate for each (N, M) pair. This is reflected in Figure3.3, and can be seen more explicitly in Figures D2

and D4. The new estimator has improved properties over the absolute-value estimator.

Lastly, I touched upon the idea that the cutoff value used to determine the success rate of a given (N, M) pair should itself be considered a parameter in the model, and care should be taken to sweep over this as well. I provide one such set of parameter sweeps in Figures D3 and D4.

3.6.2 Future Work

An extension suggested by Allen and Carroll (2001) not yet undertaken is exploring more efficient search algorithms for the grid space Θ. In part this is because I already know what a more efficient search can give us. As stated by Allen and Carroll (2001) at the end of Section 2 above, if I know which (N, M) pair provides an acceptable success rate, then I can set limits on what a more efficient search algorithm can do. If a more efficient search could allow agents to only need to examine four rules, and I know the minimum (N, M) pair which provides an acceptable success rate is $(10, 10)$, then I know it must take 10*10*4 = 400 periods to search this space. Of course, if I can**combine** highly efficient search with social learning – which I did not do here – then those rule-explorations could be spread out over the population of agents and solved in parallel.

This, then, is are some of the immediate next steps in this research program:

1. Explore the possibility of a search algorithm even more efficient than that of Chapter 2,

2. Explore different network topologies beyond the fully connected network, and

3. Work to more fully thoroughly understand the relative-value estimator.

I am optimistic that bringing these three elements together can result in a more realistic social network structure which still allows consumers to estimate a near-optimal linear consumption rule in reasonable time. In addition to standard optimization routines such as hill-climbing, I am interested in using the Particle Swarm Optimization (PSO) as a means of consumer exploration of theΘ space, as I believe it may have an attractive and

intuitive interpretation with agents as the particles. With respect to network topologies and information transmission, Carroll (2005) explores some preliminary models of inflation-expectation transmission on a network. Additional extensions include allowing the S_0 values to be heterogeneous across consumers; allowing agents to only communicate their values in a statistically noisy way as in Chamley (2004), exploring wider ranges of social learning in groups (Young, 2009), easing the credit constraint, and introducing more assets (such as housing) into the consumer's problem.

These are exciting results, and I look forward to exploring them in future research.

Table 3.2: Individual Learning versus Full Connection Absolute-Value Learning versus Relative-Value Learning

Original Results with Absolute-Value Estimator

S0=1					
		M = 1	M = 10	M = 50	M = 200
N = 10	Mean Sacrifice:	0.269	0.122	0.100	0.102
	Success Rate:	0.09	0.23	0.29	0.24
	Total Periods:	4,000	40,000	200,000	800,000
N = 20	Mean Sacrifice:	0.226	0.079	0.053	0.047
	Success Rate:	0.18	0.45	0.62	0.68
	Total Periods:	8,000	80,000	400,000	1,600,000
N = 50	Mean Sacrifice:	0.187	0.058	0.036	0.024
	Success Rate:	0.26	0.58	0.76	0.91
	Total Periods:	20,000	200,000	1,000,000	4,000,000

Full Connection Social Learning with Absolute-Value Estimator

S0 = 1					
		M = 1	M = 10	M = 50	M = 200
N = 10	Mean Sacrifice:	0.441	0.238	0.177	0.145
	Success Rate:	0.11	0.31	0.23	0.19
	Total Periods:	40	400	2,000	8,000
N = 20	Mean Sacrifice:	0.349	0.139	0.096	0.08
	Success Rate:	0.15	0.50	0.67	0.74
	Total Periods:	80	800	4,000	16,000
N = 50	Mean Sacrifice:	0.261	0.075	0.039	0.032
	Success Rate:	0.28	0.61	0.85	0.94
	Total Periods:	200	2,000	10,000	40,000

Full Connection Social Learning with Relative-Value Estimator

S0 = 1					
		M = 1	M = 10	M = 50	M = 200
N = 10	Mean Sacrifice:	0.123	0.062	0.079	0.093
	Success Rate:	0.27	0.48	0.35	0.28
	Total Periods:	40	400	2,000	8,000
N = 20	Mean Sacrifice:	0.143	0.052	0.033	0.043
	Success Rate:	0.34	0.68	0.76	0.81
	Total Periods:	80	800	4,000	16,000
N = 50	Mean Sacrifice:	0.119	0.04	0.022	0.013
	Success Rate:	0.41	0.75	0.98	1.00
	Total Periods:	200	2,000	10,000	40,000

Chapter 4: The Heterogeneous-Agent Computational toolKit: An Extensible Framework for Solving and Estimating Heterogeneous-Agent Models

4.1 Introduction

The Heterogeneous-Agent Computation toolKit (HACK) is a modular programming framework for solving and estimating macroeconomic and macro-financial models in which economic agents can be heterogeneous in a large number of ways. Models with extensive heterogeneity among agents can be extremely useful for policy and research purposes. For example, Carroll (2012a), Carroll (2014b), Carroll (2014a), and Carroll et al. (2015) demonstrate how aggregate consumption and output can be heavily influenced by heterogeneity. Geanakoplos (2010) outlines how heterogeneity drives the leverage cycle, and Geanakoplos et al. (2012) applies these insights to large-scale model of the housing and mortgage markets. However the most commonly published macroeconomic and macro-finance models have very limited heterogeneity or none at all (this includes the large class of representative agent models), in large part because these are the only models which can be easily solved with existing toolkits.[1] In contrast, models with extensive heterogeneity among agents have no central toolkit and must be solved in a bespoke way. This requires a significant investment of time and human capital before a researcher can produce publishable or usable work. This results in needless code duplication, increasing the chance for error and wasting valuable research time.

The HACK project addresses these concerns by providing a set of well-documented code modules which can be composed together to solve a range of heterogeneous-agent models.

[1] Dynare is the most popular toolkit for representative-agent models. For more details see Adjemian et al. (2011).

Methodological advances in the computational literature allow many types of models to be solved using similar approaches – the HACK toolkit simply brings these together in one place. The key is identifying methodologies which are both "modular" (in a sense to be described below) as well as robust to model misspecification. These include both solution methods as well as estimation methods.

In addition to these methodological advances, the HACK project adopts modern practices from the field of software development to ease the burden of code review, code sharing, and programming collaboration for researchers dealing in computational methods. Researchers who must review the scientific and technical code written by others are keenly aware that the time required to review and understand another's code can easily dwarf the time required to simply re-write the code from scratch (conditional on understanding the underlying concepts). This can be particularly important when multiple researchers may need to work on parts of the same codebase, either across time or distance. This problem is not confined to scientific computing alone. Fortunately the software development community, and particularly the open-source community, has spent decades perfecting tools for programmers to quickly consume and understand code written by others, verify that it is correct, and proceed to contribute back to a large and diverse codebase without fear of introducing bugs. The tools used by these professional developers include formal code documentation, unit testing structures, modern versioning systems for automatically tracking changes to code and content, and low-cost systems of communicating ideas, such as interactive programming notebooks which combine formatted mathematics with executable code and descriptive content. These tools often operate in concert with one another, forming a powerful infrastructure which can greatly accelerate project development for both individuals and collaborative teams. These technical tools are not new – the HACK project simply aims to apply the best of them to scientific code in a structured way to increase researcher productivity, particularly when interacting with other researchers' code.

The project presented here is not an attempt to create new methodology either on the software development front or the research front (although I expect new methodological

contributions to emerge from the effort). Rather the HACK project brings together many well-understood and proven methodologies to bear in an easily used and extended toolkit. The rest of this paper will first outline the useful concepts I adopt from software development, with examples of each, and then demonstrate how these concepts are applied in turn to the key solution and estimation methods required to solve general heterogeneous-agent models. If the reader is a practiced and experienced programmer, he or she may wish to skip directly to Section 3 to see how these ideas are applied to the specific consumer problem employed.

The sections are organized as follows: Section 2 outlines key tools from professional software development. Section 3 discusses the theoretical problem which provides the framework for the HACK project and outline of the first example model under development in the HACK framework. Section 4 outlines key next steps and concludes.

4.2 Tools from Software Development

Before progressing to a specific example, this section provides background about the specific software tools HACK leverages. The breadth and history of software development is extensive and review of it is beyond the scope of this document. One of the most striking practices to emerge from this history, however, is open-source software development: the decentralized collaboration of many independent programmers on a single project, often with little or no immediate monetary reward. Along with fascinating theoretical questions about incentive structures, the open-source movement has spurred the development of a wide array of excellent utilities which make decentralized code development robust and efficient. These utilities are closely intertwined with those from the traditional software development world; this section briefly overviews a number of these tools from both. The next section outlines how the solution to a basic economic problem can be developed using these utilities.

There are a number of resources which delve deeply into the topics discussed in this section. For an excellent review of many of these topics from an economists perspective, see the

unparalleled set of lectures by Sargent and Stachurski (2015), which can be accessed at the time of this writing at the Quant-Econ webpage. The Python programming language is the primary language used for development of HACK; many resources related to this language can be found on the primary Python webpage. Quant-Econ is an excellent introduction to Python for economists, as is Sheppard (2014). Aruoba and Fernández-Villaverde (2014) provide a nice comparison of many programming languages for computational economics, including Python.

4.2.1 An Aside on Speed

Python is an interpreted scripting language and at inception was many hundreds or thousands of times slower than compiled languages such as C++. As the scientific community adopts Python, a number of projects have emerged which allow Python to be compiled. At the time of this writing, there are a number of options for accelerating Python code. This is reflected in Aruoba and Fernández-Villaverde (2014), specifically their Table 1. The authors compares a number of programming languages against C++ for a loop-intensive task. When sorted by relative time against the fastest C++ implementation, Python occupies the fastest two spots which are not other C++ or FORTRAN.[2] This is not a definitive illustration of the speed capabilities of Python, as there are many caveats which must be considered in the problem setup and execution (as noted by the authors themselves). However it does serve to illustrate that Python is capable of very high speeds when compiled. Furthermore, even aside from compilation, when Python is vectorized using the major numerical libraries, NumPy and SciPy, all vectorized calculations are executed in optimized, compiled C and FORTRAN.

The one caveat in in order regarding Python speed. Object-oriented programming structures in Python may prove difficult to compile easily, and extensive computations on

[2]The first five spots in the relative ranking are occupied by different compiler implementations of C++ and FORTRAN. The 6th and 7th ranks are occupied by two of the most popular Python compilers, Cython and Numba, respectively, which are 1.41 and 1.65 times slower than the fastest C++ implementation. Notably, two C++ implementations are 1.38 times slower than the fastest, and one of the two FORTRAN implementations is 1.30 times slower than the fastest C++ implementation. That is, the fastest Python implementation is only about 3% slower than two of the three C++ implementations.

class-based objects may impose significant speed penalties. There are a number of ways around this. The simplest is to write code which does not require class structure: simple functional libraries. This is the approach HACK takes. This allows individual functions to be accelerated via vectorization or compilation, maximizing speed potential. If a class structure cannot be avoided, the accelerated functions can be called directly by members of the class, inheriting much of the speed advantages for the compiled code.[3] Finally, the HACK library written as a functional library versus a class-based system allows easy translation into additional languages if desired. Julia is a promising target for such an effort; see Sargent and Stachurski (2015) and their accompanying website for more information on the Julia language applied to economic problems.

4.2.2 Documentation

Good documentation is the key to communication between two programmers, whether between two distinct individuals or with oneself over time. In Python, as in many scripting languages, strings written on the first line after a function declaration are automatically employed as system documentation. Two popular style guide for Python documentation are found in the Google Style Guide and the Python Enhancement Proposals (PEP) system: PEP 8 and PEP 257. HACK currently uses a slight variation on the PEP 257 style guide. We illustrate this with a trivial example of a Python documentation string for a CRRA utility function. The special function documentation here is enclosed in triple quotes, which in Python sets off a multi-line string. This special function documentation is called the "docstring:"

[3]Note that if the speed advantage of the individual function comes from vectorization versus compilation, the most gain may actually be achieved by simply copy-and-pasting the function contents into the class method. See Sheppard (2014), Chapter 23, for an excellent overview of Python performance and code optimization.

```
def utility(c, gamma):
    """
    Return constant relative risk aversion (CRRA) utility of consumption "c"
    given risk aversion parameter "gamma."

    Parameters
    ----------
    c: float
        Consumption value.
    gamma: float
        Risk aversion, gamma != 1.

    Returns
    -------
    u: float
        Utility.

    Notes
    -----
    gamma cannot equal 1. This constitutes natural log utility; np.log
    should be used instead.
    """
    u = c**(1.0 - gamma) / (1.0 - gamma)  # Find the utility value of c given gamma
    return u                              # Return the utility value
```

More traditional in-line code comments may of course still be employed using the hash symbol ("#"), as can be seen in the final two lines of code above. The normal in-line comments seen above will not be included in the special function documentation, as they do not occur on the first line after the function declaration.

The special function documentation in the docstring is now employed in the formal language help system. If I query the language help files for this function I will get the following results. Note that my documentation written above now appears under the label "Docstring:"

```
>>> utility?             # The question-mark queries the help file
Type:        function
String form: <function utility at 0x7f96b473e6e0>
File:        ~/workspace/HACKUtilities/<ipython-input-20-2bbd1323015e>
Definition:  utility(c, gamma)
Docstring:
Return constant relative risk aversion (CRRA) utility of consumption "c"
given risk aversion parameter "gamma."

Parameters
----------
c: float
    Consumption value.
gamma: float
    Risk aversion, gamma != 1.

Returns
-------
u: float
    Utility.

Notes
-----
gamma cannot equal 1. This constitutes natural log utility; np.log
should be used instead.
```

In addition to traditional code documentation described above, which is included directly in the code rile, notebook-style interfaces similar to those in Mathematica and Maple have been developed for Python and a number of other languages. Programmers, including scientific programmers, have begun using these notebooks to directly communicate the ideas behind executable code to one another via html output from these notebooks, displayed in webpages. The HACK project uses Jupyter, a language-agnostic, browser-based notebook system spun off of the the IPython project. An example from the Jupyter homepage can be seen in Figure 4.1.

These notebooks can embed TeX-style mathematics typesetting alongside text and code for highly expressing scientific programming vignettes.

4.2.3 Unit Testing

Many programs are composed of a number of small functions which accomplish specific tasks. Testing at the individual function level is key to ensuring that the overall program executes correctly. This is all the more important for scientific computing, where a mistake deep in the code (eg. with a numerical approximation function) may be extremely difficult to

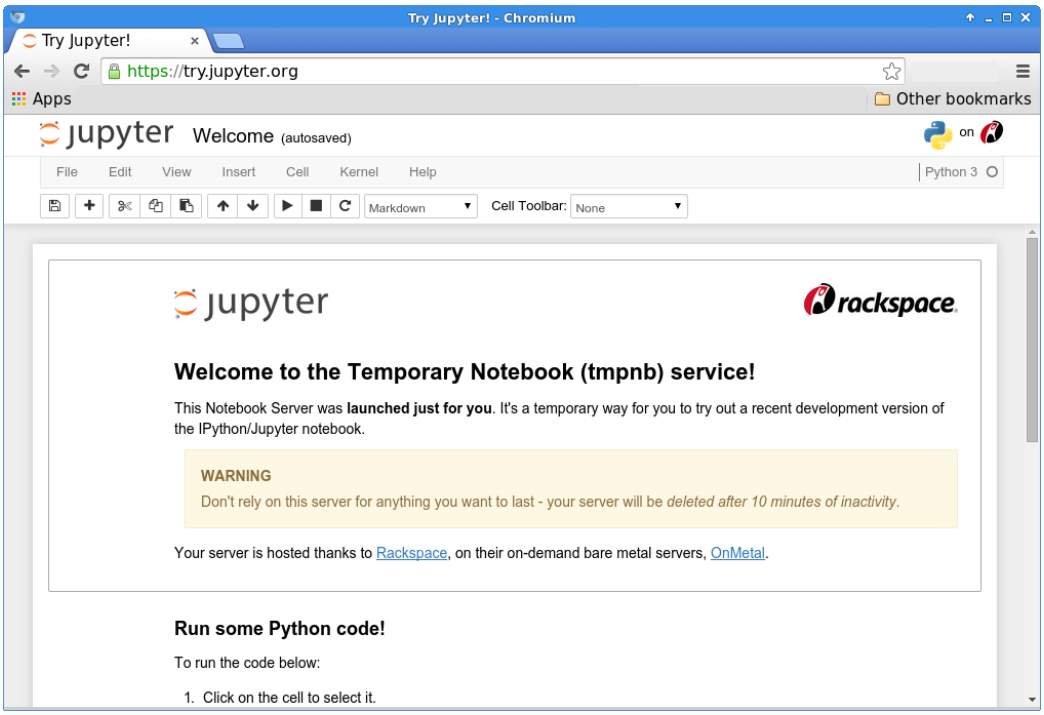


Figure 4.1: Jupyter Browser-Based Notebook

track down. Unit testing is the formal practice whereby each individual function is directly bundled with a set of tests. Each test tests a specific input and output pair, examining both "success" and "failure" states. For example, a log utility function should return a specific known value for a particular risk aversion and consumption value, and should fail with a particular error if it is presented a negative consumption value. Unit tests serve multiple purposes in code: they cover a wide range of "reasonable and representative" input and output values, and also act to conceptually illustrate when a bit of code is very complex. If it is very difficult to test a "small unit" of code, that code *may* be best decomposed into smaller and more specific-purpose functions.[4]

In scientific programming, this can serve an additional purpose: peer review of code. Uncovering bugs in code, even one's own code, can be notoriously difficult. This is many times more true when one is examining the code written by another. Thus scientific peer

[4]Note that there is a tradeoff between performance and decomposition of code into smaller and smaller units. This is discussed in Sheppard (2014).

review of code is nearly prohibitively costly, and very difficult to undertaken in a structured fashion. Unit testing can ease the burden of scientific code review in at least two ways. First, it can aid documentation in immediately outlining simple examples of code execution. Second, it can outline the pitfalls and testing procedures a reviewer may want to undertake to ensure that the code is correct. Instead of starting with a "blank page," a reviewer can take the unit tests written by the original author, run them, and then (assuming they all pass), examine the tests to see if any particular cases appear to be excluded. If so, the reviewer can use the unit tests as a template to quickly write another test case and run that as well. This can greatly accelerate both the verification of work done, as well as new testing of the code, all in a well-established and minimally costly framework.

In Python there are two built-in ways to write tests for a function: internally to the documentation, in a "doctest," and externally in a more formal unit testing framework, "unittest." Using the utility function defined earlier above, I add a doctest to the end of the function documentation (removing earlier documentation for brevity). The tests are denoted by the triple right-caret under the heading denoted "Tests." The appropriate output of the test is denoted in the line directly below the caretted line, and I will use the doctest library to run these tests. First the code definition:

```
def utility(c, gamma):
    """
    Return CRRA utility of consumption "c" given risk aversion parameter "gamma."

    ...(excluded for brevity)...

    Tests
    -----
    Test a value which should pass:
    >>> utility(1.0, 2.0)
    -1.0

    Test a value which should fail:
    >>> utility(1.0, 1.0)
    Traceback (most recent call last):
        ...
    ZeroDivisionError: float division by zero
    """
    return( c**(1.0 - gamma) / (1.0 - gamma) )
```

I save this code in a file called "utility.py." The code file now constitutes a Python module, and we use the doctest library to execute the tests embedded in the doctring. I

execute the following code:

```
>>> import utility      # Import the new utility module
>>> import doctest      # Import built-in doctest module
>>>
>>> doctest.testmod(utility, verbose=True)  # Execute doctest on utility
Trying:
    utility(1.0, 2.0)
Expecting:
    -1.0
ok
Trying:
    utility(1.0, 1.0)
Expecting:
    Traceback (most recent call last):
        ...
    ZeroDivisionError: float division by zero
ok
1 items had no tests:
    utility
1 items passed all tests:
   2 tests in utility.utility
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

The two tests passed. A contributor or a reviewer can quickly run these tests on new code, and quickly add new tests if needed. More complicated tests can be executed with the unittest framework, which is not discussed here in depth.

4.2.4 Language-agnostic, Human-Readable Data Serialization

Consider the following scenario: a researcher wants to replicate a computational model. After endless work and testing, the results between two codebases simply cannot be reconciled. Many hours are spent hunting for bugs until it is finally discovered that the problem is not in the code, but rather in a small mistake transcribing parameter settings. For some (most?) this can be an all-too-familiar experience.

One way to avoid this is using the exact same parameter settings file for all possible code-bases. One language-independent file is used to store all parameters and calibration settings for a model.[5] A replication of a particular model can use that single parameter file to confidently reproduce results across implementations. The parameter file should be easily readable by a human, as this is one more place mistakes may occur and the easier

[5]Not including the data for fitting the model – this may easily be very large and is stored separately.

to double-check, the better. Calibration of course may require many different types of data objects contained together in a single setting – floating point numbers, strings, booleans, even vectors or arrays of values. Flat-file data formats such as CSV are not flexible enough to handle all these types well. Fortunately, modern software developers have already addressed this problem with a number of options. The HACK project uses JSON (JavaScript Object Notation), a data structure somewhat analogous to simplified XML. The contents of a small JSON file may look like the following. Note the ability to include vectors, strings, and boolean values in the same file:[6]

```
{
"rho":   3.0,
"beta":  0.99,
"R":     1.03,
"liquidity_constraint": true,
"interpolation_type": "linear",
"psi_sigma": [0.001, 0.001, 0.001,
    0.28, 0.27, 0.27, 0.26, 0.26,
    0.25, 0.24, 0.23, 0.22, 0.21],
"Gamma":  [ 1.0,   1.0,   0.7,
1.01,  1.01,  1.01,  1.01,  1.01,
1.025, 1.025, 1.025, 1.025, 1.025],
}
```

The HACK project uses a single JSON file to store parameters and calibration values which can be used across multiple implementations of a model, even in multiple languages. For example, the same JSON file can be read by both a MATLAB and Python implementation of the same model.[7]

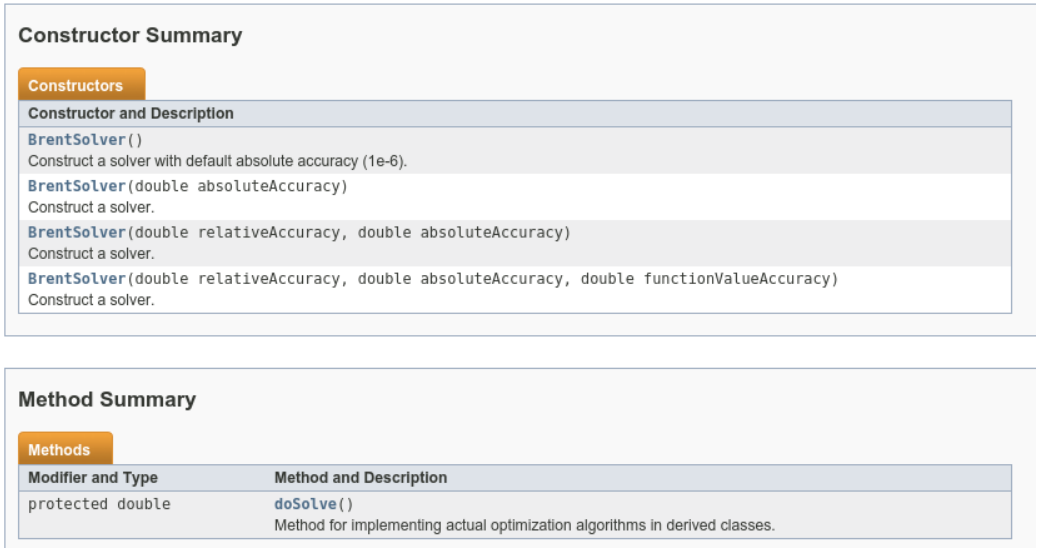
4.2.5 Application Programming Interface (API)

When contributing a module or a function to a larger code library, a programmer needs to know how this function or module fits into the overall framework of the codebase. A strict specification of variable inputs and outputs for a function communicates this information. Any large computational project with multiple developers can benefit from such a description, formally termed the the Application Programming Interface (API). This is in fact

[6]The values used in the examples in this paper are illustrative and not used for a particular estimation exercise, unless otherwise noted.

[7]A file which reads in the parameters and sets up the environment will of course be required for each language, and the researcher must be careful to treat parameters equivalently in this setup step.

simply another form of language documentation, but one that is aimed at programmers for extending or using a codebase. A clear example of a programming API for a large library of scientific code can be seen in the documentation for the Apache Math Commons Library.[8] Figure 4.2 displays an excerpt of an example API for the Brent Solver optimization routine.



Constructor Summary

Constructors

Constructor and Description
`BrentSolver()` Construct a solver with default absolute accuracy (1e-6).
`BrentSolver(double absoluteAccuracy)` Construct a solver.
`BrentSolver(double relativeAccuracy, double absoluteAccuracy)` Construct a solver.
`BrentSolver(double relativeAccuracy, double absoluteAccuracy, double functionValueAccuracy)` Construct a solver.

Method Summary

Methods

Modifier and Type	Method and Description
`protected double`	`doSolve()` Method for implementing actual optimization algorithms in derived classes.

Figure 4.2: Apache Math Commons API Excerpt for Brent Solver

The API for this Java implementation of the Brent Solver serves two purposes: first, it communicates the basic requirements for the methods inputs and outputs. Second, however, it also instructs any programmers who wish to extend the code as to the structure similar code must take.[9]

In Python an API can be formally or informally defined in a number of ways, as determined by the needs of a project. The HACK project forms a very simple API for the codebase, organized around the modules required to run a partial-equilibrium or general-equilibrium estimation. Specifically, the simple API employed by the HACK defined by the

[8] Apache Commons Math is a library of lightweight, self-contained mathematics and statistics methods addressing common problems and solutions not available in the Java programming language. See http://commons.apache.org/proper/commons-math/ for more information.

[9] For this particular example, there are more extensive details in the BaseUnivariateSolver API, here: https://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/analysis/solvers/BaseUnivariateSolver.html

main functions and data structures employed in the Estimation module, which is displayed and discussed in greater detail below. To extend the HACK library the user must replace or otherwise replicate these main functions in the Estimation module.

An further use of APIs is to define an interface between a programming language and a particular dataset. This second use of APIs, the database use, is just as important as its usage in organizing code.[10] Given the vast differences in different microeconomic database structures, this is very difficult utility to create for broad use. A preliminary version is created in the SetupEmpirical HACK module, also discussed below. This module both organizes the empirical data against which the synthetic simulation data is to be compared, and defines a function which takes raw synthetic simulation data and organizes it to be directly comparable to the empirical data.

4.2.6 Version Control

An essential tool in distributed software development is a system which can automatically archive versions of code, as well as allow the merging of changes to a document by two different programmers. Such a system is known as a version control system. The HACK project uses the Git version control system, and uses the popular online repository service Github to archive its codebase. Chacon and Straub (2014) is an excellent reference for version control in general and Git and Github in particular. Github allows code to be posted to a single online repository which tracks previous versions. A repository is copied to the computer of each contributor, and the central code source on Github may be kept private, accessible only to select users, or made widely available to the general public. Github provides a number of services on top of the pure repository service, including a simple wiki space, a space for a static website, and simple one-off repositories called "Gists," which allow the quick public or private posting of a variety of content, including Jupyter notebooks.[11]

[10] Organizations such as the Open Economics Working Group may provide a unified approach for public economic datasets.

[11] See this Github blogpost, "GitHub + Jupyter Notebooks = <3", which explicitly outlines the use of Github for sharing notebooks.

4.2.7 Bringing It Together: Reproducible Research

Many of the tools above are used to create research which can be immediately reproduced, even entirely in a web browser. This gallery of interesting IPython Notebooks outlines a number of research projects which combine code, discussion, data visualization, and descriptive mathematics to make science as transparent and reproducible as possible. For example Ram and Hadany (2015) reproduce a section of their work in an IPython notebook, which can be found here, and excerpt of which can be seen in Figures 4.3, 4.4, 4.5.

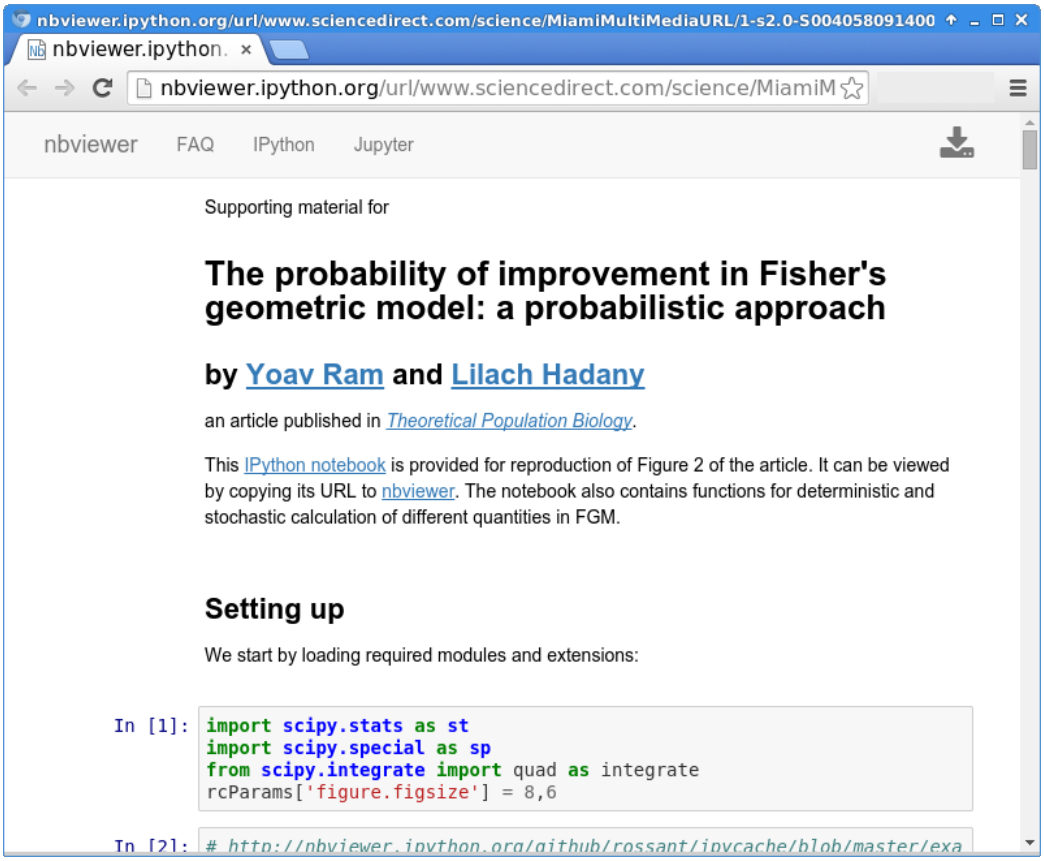


Figure 4.3: Ram and Hadany (2015) Notebook Excerpt 1

Many additional examples of reproducible research are available in the gallery noted above.

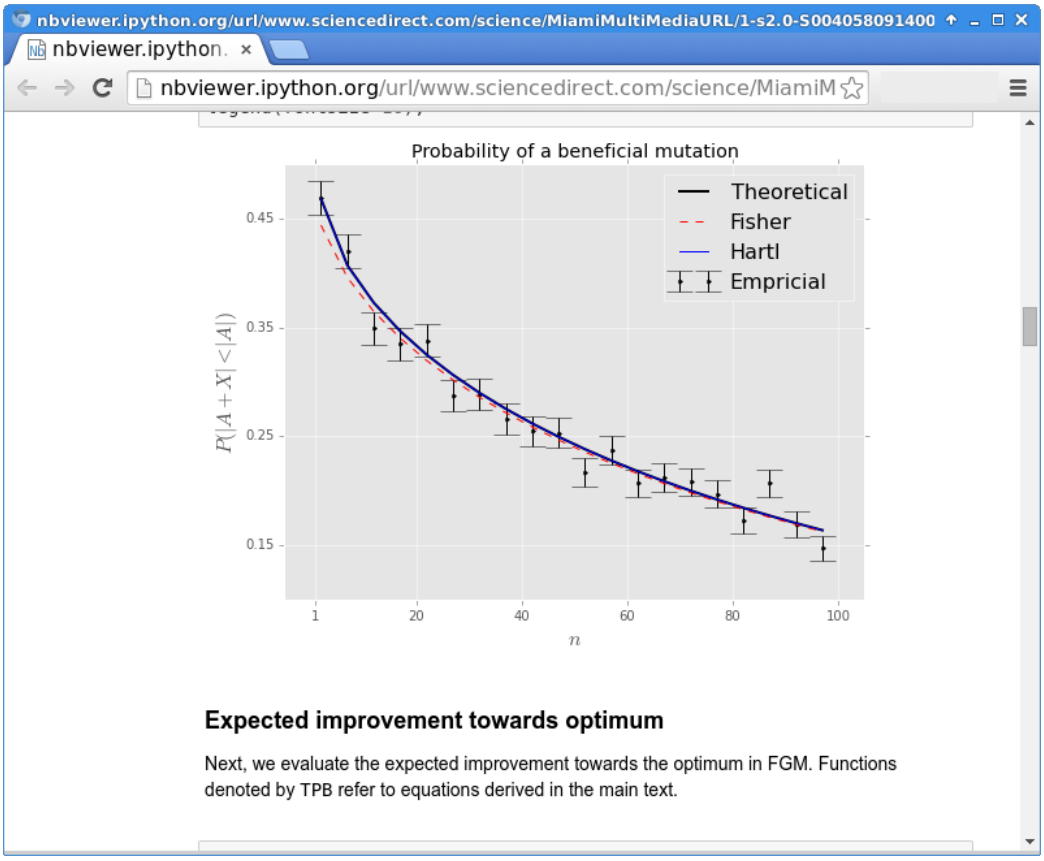


Figure 4.4: Ram and Hadany (2015) Notebook Excerpt 2

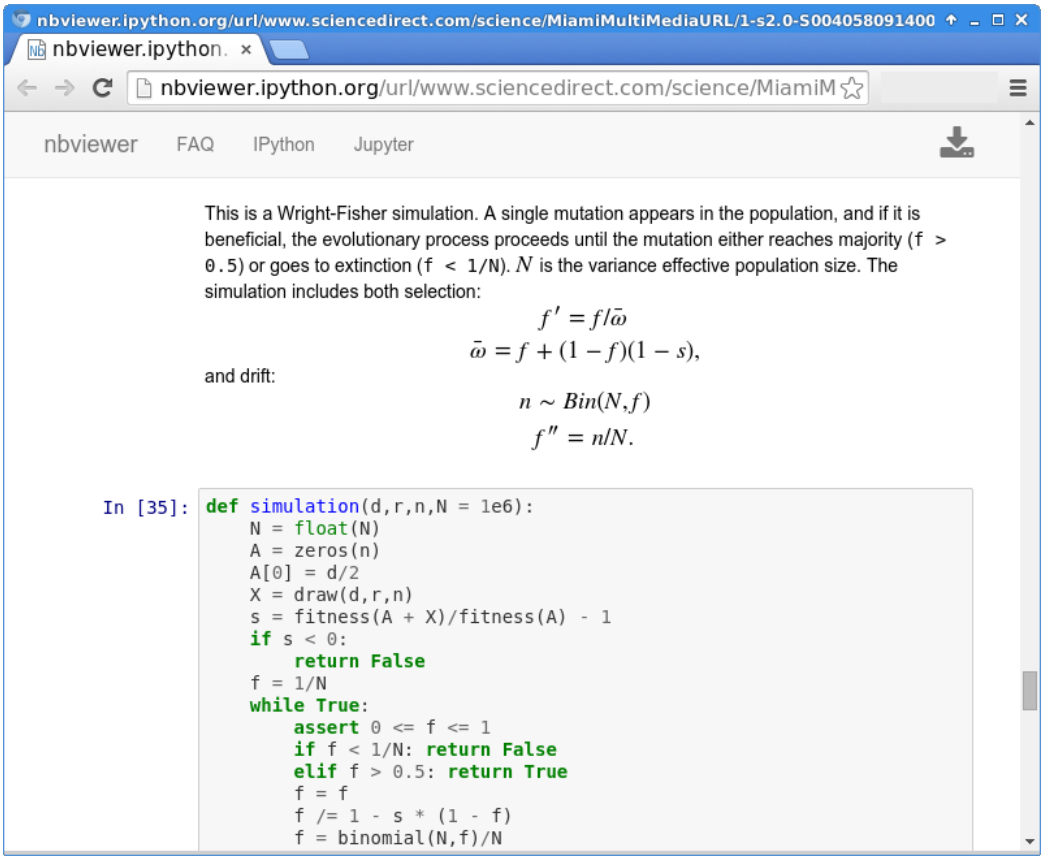


Figure 4.5: Ram and Hadany (2015) Notebook Excerpt 3

4.3 Methodological Framework

The foundational agent for the HACK toolkit is the microeconomic rational consumer. The agent's problem is stated as a dynamic stochastic optimization problem which is solved via dynamic programming. Given the solution method and appropriate data, the model is then estimated via Simulated Method of Moments (SMM), with standard errors obtained via the bootstrap. The key is to write the code such that there is a logical division between elements of the solution method. The ideal solution method decomposition should allow the various modules of the code to be agnostic to one another – if one module is replaced by a different module, which simply takes the same appropriate inputs and outputs, the solution works as before. To use a software term, the HACK project defines an API (Application Programming Interface) which instructs the user in how different solution modules communicate with one another, regardless of what they do internally. I call an element of the solution method "modular" is this is particularly easy to do. For example, as described further below, the Simulated Method of Moments estimation procedure can be robust in this way – under broad conditions, it can be applied to a very wide range of dynamic decision models[12].

One final note before proceeding. This modular approach aligns well with the authors' strategy of implementing non-optimal behavior as a departure from an already-established optimizing framework. If agents are to learn, the first straightforward extension is to learn in about some element of the solution method – eg. learn expectations or learn the optimal policy. If an agent makes mistakes, there are clear places to implement these mistakes in the optimizing framework: in the expectation function, in the law of motion, in following the optimal policy. In the codebase of HACK, the optimal problem is developed first as the framework upon which further extensions are hung. Implementing a non-optimizing solution thus involves augmenting or extending a bit of the baseline code.

This rest of this section outlines the basic optimization problem and solution method

[12]The choices represented in HACK are not the only modular solution methods which may be used, but rather a baseline. If you have a favorite solution method which you believe is robust and modular as described here, you are encouraged to contribute!

which forms the foundation of the HACK framework. The solution method is decomposed into a few major conceptual parts, which are implemented as modular libraries in HACK. Additional unimplemented solution methods are discussed. At each stage, the modular nature of the methods are noted.

4.3.1 A Basic Partial-Equilibrium Example

Consider the following finite-horizon consumption-under-uncertainty problem.[13] At time $T+1$, the consumer dies with certainty. The problem is to allocate consumption appropriately from $t=0$ to $t=T$. The full problem from Carroll (2012b) is:

$$\max_{\{\mathbf{c}_{t+j}\}_{j=0}^{\infty}} \mathbb{E}_t \left[\sum_{j=0}^{T-t} \beta^j u(\mathbf{c}_{t+j}) \right]$$

$$s.t.$$

$$\mathbf{a}_t = \mathbf{m_t} - \mathbf{c_t}$$

$$\mathbf{b}_{t+1} = \mathbf{a_t}\mathsf{R}$$

$$\mathbf{p}_{t+1} = \mathbf{p_t}\Gamma\psi_{t+1} = \mathbf{p_t}\Gamma_{t+1}$$

$$\mathbf{m}_{t+1} = \mathbf{b}_{t+1} + \mathbf{p}_{t+1}\xi_{t+1}$$

$$\mathbf{m}_0 \; given$$

where

- $\mathbf{a}_t$ is end-of-period assets,
- $\mathbf{m}_t$ is beginning-of-period total market resources ("cash on hand"),
- $\mathbf{c}_t$ is consumption in period t,
- R is a constant return factor on assets, $\mathsf{R} = (1+r)$,

[13] See Carroll (2012c) for much more detail on this style of problem.

- $\mathbf{p}_t$ is permanent non-asset income,
- Γ is a constant permanent income growth factor,
- ψ_t is a mean-1 iid permanent shock to income, and
- ξ_t is a mean-1 iid transitory shock income, composed as

$$\xi_t = \begin{cases} 0 & with\ prob\ \wp_t > 0 \\ \frac{\theta_t}{\not\wp_t} & with\ prob\ \ \not\wp_t \equiv (1 - \wp_t) \end{cases}, \text{where}$$

- $\wp_t$ is a small probability that income will be zero
- θ_t is a mean-1 iid shock transitory to income

This setup can describe a wide range of consumer circumstances, including retirement and fixed income over final years of life.

The utility function $u(.)$ is of the Constant Relative Risk Aversion (CRRA) form with risk-aversion parameter rho:

$$u(c) = \frac{c^{(1-\rho)}}{1-\rho}.$$

As in Carroll (2012b), this problem can be normalized by permanent income $\mathbf{p}_t$ to produce a simplified version of the full problem, with a reduced number of state variable. The bold symbols used above indicate non-normalized variables, while the regular non-bold symbols used below indicate variables normalized by permanent income. The normalized problem can be written in Bellman form:

$$v_t(m_t) = \max_{c_t} u(c_t) + \beta \mathbb{E}_t \left[\Gamma_{t+1}^{1-\rho} v_{t+1}(m_{t+1}) \right]$$

$$s.t.$$

$$a_t = m_t - c_t$$

$$b_{t+1} = \left(\frac{\mathsf{R}}{\Gamma_{t+1}} \right) a_t = \mathcal{R}_{t+1} a_t$$

$$m_{t+1} = b_{t+1} + \xi_{t+1}$$

$$m_0 \; given$$

or simplified further:

$$v_t(m_t) = \max_{c_t} u(c_t) + \beta \mathbb{E}_t \left[\Gamma_{t+1}^{1-\rho} v_{t+1}(m_{t+1}) \right]$$

$$s.t.$$

$$m_{t+1} = \mathcal{R}_{t+1}(m_t - c_t) + \xi_{t+1}$$

$$m_0 \; given$$

4.3.2 The Solution Method

The general solution method is as follows: in the final period T, the value function in the following period is $v_{T+1}(m) = 0 \; \forall_m$, and the value function in period T is simply $v_T(m) = u(m)$. This makes the problem in period $T-1$ straightforward to solve numerically for both the consumption function and value functions $c^*_{T-1}(m)$ and $v^*_{T-1}(m)$:

$$c^*_{T-1}(m) = \underset{c\in[0,\bar{m}]}{argmax}\ u(c) + \beta\mathbb{E}_{T-1}\left[\Gamma_T^{1-\rho}u(\mathcal{R}_T(m-c)+\xi_{t+1})\right]$$

and

$$v^*_{T-1}(m) = u(c^*_{T-1}(m)) + \beta\mathbb{E}_{T-1}\left[\Gamma_T^{1-\rho}u(\mathcal{R}u(c^*_{T-1}(m))\right]$$

where $\bar{m}$ is a self-imposed liquidity constraint.[14]

With these numerical solutions in hand, the solution method is now simply recursive: step back one more period to $T-2$ and solve for optimal consumption and value functions using $c^*_{T-1}(m)$ and $v^*_{T-1}(m)$. This process can be continued back until the first period $t=0$. This solution process is outlined in greater detail in Carroll (2012b).

4.3.3 The Estimation Method

Denote the behavioral parameters β, ρ, (discounting and risk aversion, respectively) as

$$\phi = \{\beta, \rho\}$$

and denote the structural problem parameters as

$$\varrho = \{\varrho_t\}_{t=0}^T, \text{where}$$

$$\varrho_t = \{\Gamma, \psi_t, \xi_t, \wp_t, \theta_t\}, \forall_t.$$

Given an arbitrary behavioral parameter set $\phi = \{\beta, \rho\}$, and choosing the values and data-generating processes for the structural problem parameters ϱ to match consumer experiences in the PSID, I can solve for the set of consumption functions which are optimal under these conditions, $\{c^*_t(m)\}_{t=0}^T$.

[14] Carroll (2012b) demonstrates the reasoning behind this derivation. In a model with positive probability of a zero-income event, $\bar{m} = m$.

With these consumption functions now in hand, I can use the calibrated parameters ϱ to generate N different simulated consumer experiences (vectors of income shocks) from $t = 0, 1, ..., T$. Applying the consumption functions $\{c_t^*(m)\}_{t=0}^T$ to this set of simulated experiences generates a N-sized distribution of simulated wealth holdings for all t. The moments of these cross-sectional distributions of wealth can then be compared to the equivalent moments in appropriately constructed empirical data from the Survey of Consumer Finance (SCF). I form the following objective function, which compares population median between empirical wealth-to-income ratio from the SCF and its simulated equivalent:

$$\varpi_\varrho(\phi) \equiv \sum_{i=1}^{N} \omega_i |\zeta_i^\tau - \mathbf{s}_\varrho^\tau(\phi)|.$$

Here $\varpi_\varrho(\phi)$ represents the objective value for the distance between medians of the two populations, the synthetic population variables represented by (s) and the empirical population variables represented by ζ (see Carroll (2012b) for more discussion of the form of this objective function for population moments). The index i indicates individual observations in the empirical data, each of which has a population weight ω_i (required in the SCF due to oversampling of particular sub-populations). Each individual i in the empirical data has observations at the age-group frequency, τ. The variable $\mathbf{s}_\varrho^\tau(\phi)$ is the median of the simulated data for age group τ, under calibration ϱ, using the parameters $\phi = \{\beta, \rho\}$. Once this value has been constructed as a function of ϕ, the estimation occurs by simply finding the minimal ϕ value numerically:

$$\min_{\phi} \varpi_\varrho(\phi).$$

This is accomplished in code by simply handing the expression $\varpi_\varrho(\phi)$ to a numerical minimization process. The standard error on the resulting estimation of $\{\beta^*, \rho^*\}$ is found by bootstrapping the empirical data and repeating the above estimation process a number

of times, $N_{bootstrap}$.

4.3.4 Modular Solution and Estimation in HACK

The solution and estimation method described for the basic problem above can be decomposed into the following steps, each of which is written as a module in Python. Each module is documented, tested, and brought together in IPython notebook "vignettes" to demonstrate their use, and finally brought together in a simple interface to effect model solution and estimation. Extending the partial-equilibrium toolkit corresponds to writing a new version of a specific set of functions in each of the basic modules, using the exiting code and vignettes as examples and guides.

The major conceptual solution and estimation components for the partial-equilibrium problem are:

- parameter definition
- setup of data and data/simulation comparison
- expectations formation and calculation
- value and policy formation
- simulation of population experience under a particular policy
- estimation of parameters using SMM and bootstrapping

These parts together form the basis of the partial-equilibrium portion of HACK. Code is divided into the following primary modules, corresponding to the solution method breakdown noted above:

- SetupParameters.py
- SetupEmpirical.py
- HACKUtilities.py
- SolutionLibrary.py
- SimulationLibrary.py

- Estimation.py

This section examines these basic modules and outlines the process by which the library can be expanded to include additional models. Work is underway to build out the general-equilibrium portion using the approach implemented in Carroll et al. (2015). Namely, the following additions will be made:

- price-finding via market clearing
- rational expectations via the Krusell-Smith (1998) algorithm.

The rest of this section outlines the main contents of the five modules noted above. Each is illustrated with pseudo-code headers and content as needed. For each module, the primary functions which need to be modified to extend the baseline model are identified and discussed.

The module discussion begins with the final Estimation module listed above, as this module clearly outlines the specific functions and intermediate data structures which must be overwritten to extended the basic HACK framework to solve and estimate another model.

Estimation.py

The final and central module in HACK is the Estimation module. This brings all the others together to solve, simulate, and estimate the preference parameters for the basic consumption-under-uncertainty problem outlined above. In the pseudo-code below, all the major functions which are necessary for the operation of the HACK project are outlined. To change the baseline model, one only needs to change the five major functions imported from other modules and used in the Estimation module. These five functions and their definitions comprise the programming API for the HACK framework:

- SimulationLibrary:
  - *create_income_shocks_experience*
  - *find_wealth_hist_matrix*

- SolutionLibrary:
  - *init_consumer_problem*
  - *solve_consumption_problem*
- SetupParameters:
  - *find_simulated_medians*

The module first imports all necessary HACK modules, executes the primary functions from each, creates the SMM objective function $\varpi_{\varrho}(\phi)$, and executes a single minimization:

$$\min_{\phi} \varpi_{\varrho}(\phi).$$

Also included in this module is a bootstrap function which repeats the minimization for a bootstrap sample of data, $N_{bootstrap}$ times.

A special note for the pseudo-code for Estimation.py that follows: the Python operator "**" acts to unpack a dictionary (a hash-table data storage object in Python) and use its key to associate the dictionary values with the appropriate function calls. Thus the definitions of the dictionaries "unpacked" by the ** symbols below act to keep the code clean and readable.

This code excerpt includes the pseudo-code for the calculation of the SMM objective function, *smm_objective_fxn*, which is almost entirely complete code:

```
import SetupParameters as param
import SetupEmpirical as empirical
import SolutionLibrary as solution
import SimulationLibrary as simulate

# ------ Create income shock draws ------ #
agent_shocks_matrices = simulate.create_income_shocks_experience(**param.create_income_shocks)

# ------ Initialize the consumer problem ------ #
income_distrib = solution.init_consumer_problem(**param.init_consumer_problem)

# ------ Create full collection of calibration parameters ------ #
calibrated_parameters = {"shocks":agent_shocks_matrices, "income":income_distrib}
calibrated_parameters.update(param.calibrated_parameters)

# ------ Define the SMM objective function ------ #
def smm_objective_fxn(phi, **calibrated_parameters):
    #  Solve consumption functions
    consumption = solution.solve_consumption_problem(phi, ...)
    # Run simulation and get simulated wealth holdings
    sim_m_history = simulate.find_wealth_hist_matrix(...)
    # Construct simulated medians correctly for efficient calculation...
    simulated_medians = empirical.find_simulated_medians(sim_m_history)
    # Return sum of absolute errors
    return np.dot(empirical_weights, np.abs(empirical_data - simulated_medians))


# ------ Execute a single minimization on the SMM objective function ------ #
def minimize_smm_objective(minimizer):
    # Use a numerical optimizer to find the minimum value from smm_objective_fxn

# ------ Bootstrap the minimization on the SMM objective function ------ #
def bootstrap(optimizer, param.empirical_data, param.Ndraws, **calibrated_parameters):
    # Replicate the SMM estimation a number of times, bootstrapping empirical data

# ------ Execute the Estimation ------------------------------------- #
def main():
    # Execute simple user interface for estimation options.
```

The beauty of the modular HACK structure emerges in this Estimation module. To start an extension of the baseline model, the user first identifies which of the functions in the above process must be overwritten or extended. Once this is established, the user simply walks through the codebase and makes the appropriate adjustments. Once that work is done, only minor changes are required for Estimation.py. Furthermore, the documentation, testing, and organization of all other methods are outlined in the baseline example, easing the process for the new user.

Figure 4.6 displays a screen shot from the simple command-line interface to the estimation process. The estimation module prompts the user for a selection from numerical optimization options, then asks for initial conditions. After the initial estimation, the option to bootstrap the standard errors is provided.

```
IPython 3.0.0 -- An enhanced Interactive Python.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]: execfile("Estimation.py")


Please choose a solution method by typing its letter, [Enter] to use Nedler-Mead, or q to quit:
[a] Nedler-Mead    [default]
 b  Powell.
 c  Brute search and contour plot.
a

Please enter a numerical value for initial beta guess,
or hit return to use the default value of
[0.99]
1.0

Please enter a numerical value for initial rho guess,
or hit return to use the default value of:
[4.0]
3.5
Starting Nelder-Mead with initial value (beta, rho) = (1.0, 3.5)
Optimization terminated successfully.
         Current function value: 42600.403076
         Iterations: 51
         Function evaluations: 99
Output: [ 0.92637873  2.66819023]
time to solve: 0.518002335231 min
time to solve: 31.0801401138 sec

Would you like to see the bootstrap estimates of variance?
Y/[N]
```

Figure 4.6: Simple Simulated Method of Moments Estimation User Interface

The brute force optimization option "c" will search a fixed grid for the optimal point, and produce a contour plot over the grid as part of the output. The surface for the particular parameterization used for this paper can be seen in Figure 4.7.

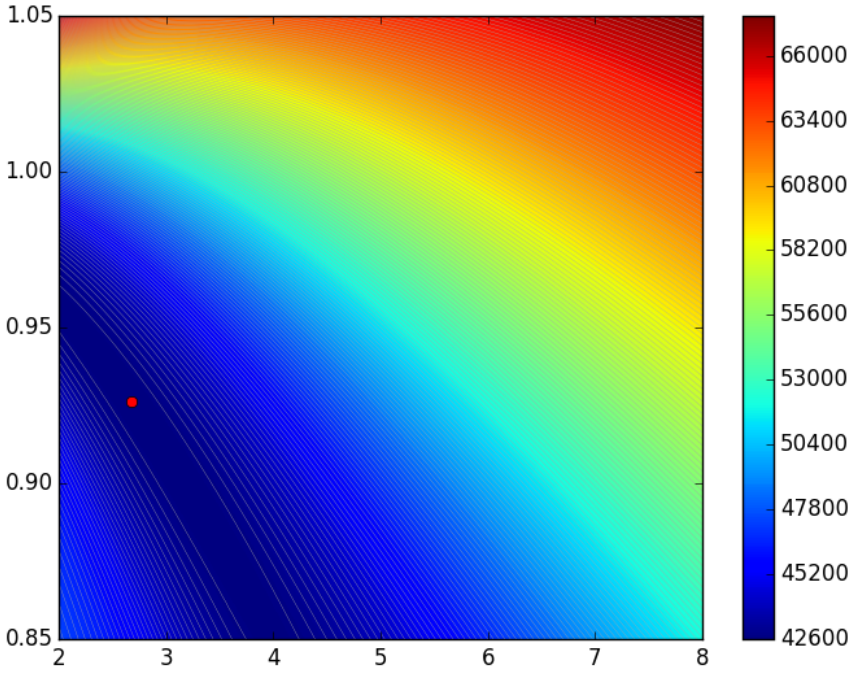


Figure 4.7: Simple Simulated Method of Moments Estimation Contour Plot

SetupParameters.py

This module is coupled with an input JSON file, which specifies the full set of calibrated values required to solve the model. The JSON file, as noted previously, is language-independent, and can be read and used by nearly any modern programming language. The HACK project has used this in particular to validate multiple-language versions of the same model, eg. between MATLAB and Python. This setup allows calibration parameters to be specified once, in a separate, easily human-readable file. Importantly, the SetupParameters also executes a key function for the estimation step: it brings in and organizes the empirical

data to be used in the estimation process,[15] and it defines a function *find_simulated_medians* which will take in simulated wealth data and organizes it to be comparable to the empirical data as stated in the SMM objective expression,

$$\sum_{i=1}^{N} \omega_i |\zeta_i^\tau - \mathbf{s}_\varrho^\tau(\phi)|.$$

When the user desires to change the estimation procedure (eg. change the empirical data or moments compared), the SetupParamters file must be changed appropriately.

This SetupParameters file is imported into any subsequent module which needs to access the parameters using the following line of code. Note that particular parameters are immediately accessible:

```
>>> import SetupParameters as param
>>>
>>> print "R =", param.R
R = 1.03
>>> print "Gamma =", param.Gamma
array([ 1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,
        0.7  ,  1.01 ,  1.01 ,  1.01 ,  1.01 ,  1.01 ,  1.01 ,  1.01 ,
        1.025,  1.025,  1.025,  1.025,  1.025,  1.025,  1.025,  1.025,
        1.025])
>>> print "Initial beta guess for SMM optimization:", param.beta_start
Initial beta guess for SMM optimization: 0.99
```

Note that the JSON file includes initial guesses for the parameters to be estimated by the Simulated Method of Moments routine. To extend the baseline HACK model, new calibrated parameter values will need to be added to this file.

SetupEmpirical.py

This module sets up empirical data, along with functions which allow moments of the empirical data to be matched to moments of the simulated data. The main functions are:

[15] This data is usually stored in a separate format than the parameters in the JSON file.

```
def find_empirical_mappings(...):
    pass

def setup_simulated_medians(...):
    pass

def def bootstrap_data_and_components(...):
    pass
```

The first function constructs mapping functions between the empirical and simulated data moments. The second function uses these mappings to organize simulated data moments to map to empirical data moments, and the final function implements data resampling procedures required for the bootstrap estimates of variance.

HACKUtilities.py

This module contains a number of utilities used by the HACK framework, including the code to implement agent expectations. Agent expectations here are implemented as a discretization of the shock-space, achieved by choosing the size of discrete points to represent the distribution, $N_{discrete}$, creating an equiprobably-spaced partition over the support, and selecting the representative point in each partition as the conditional mean of values in the partition. Each resultant point is then assigned the probability $\frac{1}{N_{discrete}}$. See Carroll (2012b) for a detailed discussion of this approach.

An example of the code which implements a mean-1 lognormal shock space is as follows:

```
def calculate_mean_one_lognormal_discrete_approx(N, sigma):
    '''
    Calculate a discrete approximation to a mean-1 lognormal distribution.

    Parameters
    ----------
    N: float
        Size of discrete space vector to be returned.
    sigma: float
        standard deviation associated with underlying normal probability distribution.

    Returns
    -------
    X: np.ndarray
        Discrete points for discrete probability mass function.
    pmf: np.ndarray
        Probability associated with each point in X.

    Test
    ----
    Confirm that returns discrete mean of 1
    >>> import numpy as np
    >>> x, pmf = calculate_mean_one_lognormal_discrete_approx(N=5, sigma=0.2)
    >>> np.dot(x,pmf)
    1.0
    '''
    mu = -0.5*(sigma**2)
    distrib = stats.lognorm(sigma, 0, np.exp(mu))

    # ------ Set up discrete approx ------
    pdf = distrib.pdf
    invcdf = distrib.ppf
    probs_cutoffs = np.arange(N+1.0)/N      # Includes 0 and 1
    state_cutoffs = invcdf(probs_cutoffs)   # State cutoff values, each bin

    # Set pmf:
    pmf = np.repeat(1.0/N, N)

    # Find the E[X|bin] values:
    F = lambda x: x*pdf(x)
    Ebins = []

    for i, (x0, x1) in enumerate(zip(state_cutoffs[:-1], state_cutoffs[1:])):
        cond_mean1, err1 = quad(F, x0, x1, epsabs=1e-10, epsrel=1e-10, limit=200)
        # Note that the *first* to be fulfilled of epsabs and epsrel stops the
        # integration - be aware of this when the answer is close to zero.
        # Also, if you never care about one binding (eg if one would like to
        # force scipy to use the other condition to integrate), set that = 0.
        Ebins.append(cond_mean1/pmf[i])

    X = np.array(Ebins)
```

Note the embedded doctest, which runs "sanity checks" on the discretization process. Tests such as these identified initial numerical errors in the discretization process due to loose default integration tolerances – a key contribution of unit testing which can help avoid many hours of bug hunting in incorrect portions of the codebase.

The key modular feature of the HACKUtilities library is that it produces, finally, a single discrete representation of the probability space faced by the consumer. As long as shocks are iid, the number of dimensions of shocks does not matter – each distribution is discretized and the joint distribution is create combinatorially from the individual discrete marginal

distributions. To create expectations, the HACKUtilities library finally produces a set of combinations of all discrete points as the support, and the corresponding combination of all discrete probabilities as the distribution. Expectations of a function f are then formed simply by the dot product of f applied to each point with the probabilities associated with all points.

If additional methods of expectations formation are desired, this is the correct module in which to develop them.

SolutionLibrary.py

The solution library contains the main definitions used in the solution method. The HACK project uses dynamic programming to determine the solution to the consumer problem, and in particular uses the endogenous gridpoints method to greatly accelerate solving for the policy function. The endogenous gridpoints method is discussed in extensive detail in Carroll (2006). This solution method takes advantage of the *end of period* consumption and value functions,

$$\mathfrak{c}(a_t) \text{ and } \mathfrak{v}(a_t)$$

which map end-of-period wealth a_t to consumption and expected value. The endogenous gridpoints method is not required for the HACK project, but it greatly accelerates the solution method and is used in the baseline HACK model. This is one example of including concrete examples of non-trivial computational "tricks" which may greatly improve a solution method. The endogenous gridpoints method may not be easy to understand upon first encounter. Thus the HACK toolkit includes it along with an IPython notebook which quickly illustrates how the process works – an interactive and executable summary of Carroll (2012b).

The key methods in the SolutionLibrary module are the following, shown only with their function headers – documentation and code details are excluded for brevity. The main functions are:

Utility functions: The CRRA utility function is defined with its first and second derivative:

```
def utility(c, gamma):
    pass

def utilityP(c, gamma):
    pass

def utilityPP(c, gamma):
    pass
```

The end-of-period consumption function for period $T-1$ and all other $t < T-1$ These are the key functions used in the endogenous gridpoints backwards induction method:

```
def gothicC_Tm1(a, rho, uP, R, beta, Gamma, psi_support, xi_support, pmf):
    pass

def gothicC_t(a, c_prime, rho, uP, R, beta, Gamma, psi_support, xi_support, pmf):
    pass
```

Initialize the consumer problem: A consumer's problem must be set up before it can be solved: the expectations support and probability mass function must be created for each period:

```
def init_consumer_problem(R, Gamma, constrained,
                          psi_sigma, psi_N, xi_sigma, xi_N, ...):
    pass
```

Solve the consumer problem: The recursive solution method is implemented by two functions: the first solves a single period in the problem ("one step back"), while the second implements the full backwards recursion, from period $T-1$ to 0:

```
def step_back_one_period(rho, beta, R, Gamma, shocks, pmf, a_grid):
    pass

def solve_total_consumption_problem(rho, beta, R, Gamma, shocks, pmf, a_grid):
    pass
```

The final function, "*solve_total_consumption_problem*," returns a list of consumption

function objects, ordered in reverse chronology (index 0 is the consumption function for period T, index 1 is consumption function for $T-1$, etc.). All other functions in the SolutionLibrary module can be thought of as supporting the final *solve_total_consumption_problem* function.

The baseline model can be extended by overwriting the *solve_total_consumption_problem* function, as well as creating/overwriting whatever additional functions are needed to support the new implementation. For example, if a model is extended with respect to solution methods – for example, if Bayesian learning is added to the agent solution method – this module is the correct place to include that extension.

SimulationLibrary.py

The simulation library implements the simulation step of the estimation process – given a reverse-chronology list of consumption functions, the functions in this library will draw an appropriate panel of shocks of size $\{T, N_{simulate}\}$, where T is total periods and $N_{simulate}$ is the total number of agents for which to simulate experiences (eg. $T = 60$ and $N_{simulate} = 10,000$).

The key methods in the SimulationLibrary module are the following, again shown only with function headers for brevity:

Main two functions: The main two functions in the SimulationLibrary module first create the complete set of shocks required to simulate agent experiences and, second, apply the consumption solution from the SolutionLibrary to this set of shocks to produce the simulated wealth panel. To extend the baseline model, these are the two major functions to overwrite, as well as the requisite helper functions. The two major functions appear as follows:

```
def create_income_shocks_experience(psi_sigma, xi_sigma, Gamma, R, p_unemploy, T, N):
    pass

def find_wealth_history_matrix(policies, state0, agent_shocks_matrices):
    pass
```

Helper functions: There are a number of "helper" functions in the SimulationLibrary

which support *create_income_shocks_experience* and *find_wealth_history_matrix*. The single task of creating and simulating the shocks is split across a number of helper functions to aid in both clarity and unit testing. Together they are used to create the final matrices of shocks needed to simulate consumption:

```
# ------- Generate permanent and transitory income shocks ------- #
def generate_permanent_income_draws(psi_sigma, N_simulate):
    pass

def generate_transitory_income_draws(temp_sigma, p_unemploy, N_simulate):
    pass

# ------ Create joint discrete distribution from independent marginals ----- #
def generate_all_combined_shocks(shocks1, p1, shocks2, p2):
    pass

# ------ Generate matrix of perm and transitory shocks of correct size ------- #
def create_shocks_matrix_1D(shocks, N_simulate, T, seed=None):
    pass

def create_shocks_matrix_1D_with_zero_income_event(shocks, N, T, seed=None):
    pass

# ------ Generate retirement income if needed ------ #
def generate_retire_income(psi_retire, xi_retire, Gamma, R, p_unemploy_retire):
    pass
```

4.4 Summary and Conclusion

The HACK project is a modular code library for constructing macroeconomic and macro-financial models with heterogeneous agents solving portfolio decisions under uncertainty. Portfolio choice under uncertainty is central to nearly all academic models, including modern DSGE models (with and without financial sectors), models of asset pricing (eg. CAPM and C-CAPM), models of financial frictions (Bernanke et al., 1999), and many more. Under the right assumptions many of these models can be solved by aggregating agent decision-making and employing the representative agent, with standardized computational frameworks for solving these models. However when individual agents look very different from one another - for example, different wealth levels, preferences, or exposures to different types of shocks - assumptions required for aggregation can quickly fail and a representative agent may no

longer be appropriate. Code to solve these models tends to be bespoke and idiosyncratic, often reinvented by different researchers working on similar problems. This needless code duplication increases the chance for errors and wastes valuable researcher time.

Researchers should spend their valuable time producing research, not reinventing the wheel when it comes to computational tools. The goal of the HACK toolkit is to ease this burden by providing a simple and easily extensible framework in which a few common models are solved, and clear documentation, testing, and estimation frameworks provide guidance for new researchers to develop their own work in a robust and replicable manner. The final goals of the project are to create a collaborative codebase which can serve both researchers and policymakers alike, employing the best of modern software development tools to accelerate understanding and implementation of cutting edge research tools. The solution methods employed in HACK are not the only methods available, and those who have additional methodological suggestions are strongly encouraged to contribute! Increasing returns to production is one of the few "non-dismal" possibilities in economic thought – I hope to capture this feature of code production in the HACK framework. Key next steps include finalizing the general-equilibrium HACK modules, identifying additional baseline models to replicate in HACK, and encouraging a new generation of students to learn from, use, and contribute to the collaborative construction of heterogeneous-agent models.

Chapter 5: Conclusion

Economics has encountered a crisis of methodologies following the Financial Crisis and Great Recession. As awareness grows of the need for increased heterogeneity and complexity in models, there is a countervailing push to maintain some form of dynamic optimizing behavior, which is generally intractable as heterogeneity and complexity increase. Agent-based modeling has only offered a partial solution – the ability to model complex environments and situations is there, but a widely accepted method of tractable bounded rationality has not appeared. In the past two decades, however, the dynamic programming literature has developed learning-to-optimize methods which generalize dynamic programming by easing the information requirement on decision makers. Agents instead learn near-optimal behavior from trial-and-error experience. This behavior produces a bounded rationality from first principles: agents still maximize familiar preferences under uncertainty, but are simply restricted from solving the fixed-point value-function calculation each period. Instead agents solve the fixed-point calculation over time, via experience.

Chapter 2 in this dissertation introduces regret learning, which is a simple learning-to-optimize algorithm which allows agents to solving their consumption-under-uncertainty problem by minimizing "regret:" after learning a value function over a set of periods, agents look back over that experience and calculate the choices which would have maximized the per-period Bellman, "given what they know now," i.e. the learned value function. This produces a new consumption function, and the process is repeated. Simulations are used to show that this converges to a neighborhood around the optimal solution, and there is extensive discussion of the theoretical framework on which regret learning is based.

Chapter 3 extends Allen and Carroll (2001)'s original agent-based model in two ways: first, I incorporate social learning into the process, and second, I introduce an new, intuitively motived estimator of the value of a simple linear consumption rule. Social learning

occurs through a simple form of information sharing. This addition retains the original model's results that consumers can consistently find an optimal rule, but lowers the time required to find such a rule arbitrarily close to the lowest possible bound. The time required to find a rule is now a function of the number of agents in the model. Furthermore, the new estimator I identify further decreases the amount of time required to find a near-optimal rule by a full order of magnitude. This estimator also opens the door for the social learning process to incorporate heterogeneous initial endowment values across agents, a welcome extension to the original model.

In Chapter 4, I present initial work on a modular and extensible toolkit for solving and estimating heterogeneous-agent partial- and general-equilibrium models. Heterogeneous-agent models have grown increasingly valuable for both policy and research purposes, but code to solve such models can be difficult to penetrate for researchers new to the topic. As a result it may take years of human capital development for researchers to become proficient in these methods and contribute to the literature. The goal of the HACK toolkit is to ease this burden by providing a simple and easily extensible framework in which a few common models are solved, and clear documentation, testing, and estimation examples provide guidance for new researchers to develop their own work in a robust and replicable manner. Using two examples, I outline key elements of the toolkit which ease the burden of learning, using, and contributing to the codebase. This includes a simple API for model solution and an API for estimation via simulation, as well as methods for bundling working code with interactive documentation. The foundational solution method I employ is Carroll (2012b), Solution Methods for Microeconomic Dynamic Stochastic Optimization Problems, written in a modular Python framework.

All code will be open and published to a public repository.

5.1 Results

The dissertation contributes to the current literature by providing a rigorous description of plausible mechanisms by which an agent can learn the solution to a dynamic stochastic

optimization problem from experience. This dissertation offers three analogies regarding how agents might learn from their own experience and the experience of others. These analogies are then made rigorous by casting them in the framework of approximate dynamic programming. Numerical simulation results then show that these mechanisms converge towards the optimal solution in distribution as experience is taken to the infinite limit.

The contribution of the learning-to-optimize literature in general and this dissertation in particular is the surprising note that agents can obtain optimal solutions from learning in real time. This is not a result which is clear from observing the first principles of dynamic optimization theory. Even when an agent only has access to a single stream of experience, as in Chapter 2 in this dissertation, and thus their own learning is highly correlated, they can nonetheless find an near-optimal policy function. To say it another way, estimating a value function from very little experience produces a highly biased value function – and yet learning a highly biased value function can still lead to learning a nearly correct consumption function.

Finally, the solution method presented in this dissertation is explicitly intended to be an engine to drive development of both agent-based models such as Geanakoplos et al. (2012) and large-scale heterogeneous-agent macroeconomic models such as Peterman et al. (2015).

5.2 Weaknesses and Future Work

The learning to optimize work described above for Allen and Carroll (2001) and others is not without potential criticism. From a more traditional economic learning perspective in which agents learn beliefs but still optimize conditional on beliefs, such as in Chamley (2004) or Evans and Honkapohja (2001), the criticism may be leveled that agents who learn to optimize are taking learning too far – essentially, there are no good reasons to carry bounded rationality beyond expectation formation and into the optimization process itself. I have personally heard this in one-on-one conversation, but even Evans and McGough (2014) observes that learning only about expectations is likely not enough.

From the agent-based side of research, the opposite charge may be made: that bounded

rationality which only approximately optimizes is still too much usage of the optimization framework. This is a balance that must always be struck carefully. One future extension of the regret learning framework takes this criticism to heart; a much simpler mechanism than the one presented here may achieve nearly the same results. That work is underway.

Finally, there are a number of limitations to the models to be presented in the preceding essays: agents learn faster than in the original Allen and Carroll (2001) setup, but still learn slowly. Agents do not take into account large one-time decisions such as buying a house or retiring, and agents do not trade assets with one another, or save money in other savings vehicles, such as the stock market or mutual funds. Thus there is no immediate way to examine general equilibrium effects. All of these shortcomings indicate future directions of research. In particular, the results of the first two chapters can be combined into a single social and individual learning model. This is the immediate next step in this research stream.

Appendix A: Policy Iteration and Optimistic Policy Iteration Proofs

The proofs in this appendix closely follow the conventions and notation in Bertsekas (2012). These are largely re-derivations of his proofs for policy iteration and optimistic policy iteration, although a number of the lemmas are proved for a more general case than he presented and following a stricter expositional style when proving by induction. The final extension of the proofs to the alternate ordering of the optimistic policy iteration steps is simple but necessary for determining the conditions under which the initial policy and value functions employed by agents will be known to converge to their optimal counterparts.

I begin by defining important notation and reviewing the mathematical essentials required for the following proofs. As noted above, the notation here differs slightly from that in the text.

Define the following. Let:

- X be a set of states with elements $x \in X$,
- U be a set of controls with elements $u \in U$,
- $U(x) \subset U$ be a "feasible set" of controls $\forall\, x \in X$
- $\mathcal{M}$ be the set of all policy functions with elements

$$\mu \in \mathcal{M} \equiv \{\mu : X \to U \; s.t. \; \mu(x) \in U(x)\}$$

- $r : U \to \mathbb{R}$ is the real-valued "reward" function,
- $J : X \to \mathbb{R}$ is the real-valued "value" function, $J \in \mathsf{R}(X)$
- $H : X \times U \times \mathsf{R}(X) \to \mathbb{R}$ is a mapping
- $w \sim W$ be a random shock distributed as distrbution W,
- $f : X \times U \times W \to X$ be a transition function, and
- $\beta \in (0, 1)$ be a discount factor.

Define the mapping T as:

$$(TJ)(x) = \sup_{u \in U(x)} H(x, u, J) \ \ \forall x \in X$$

where we use the shorthand $(TJ)(x) \equiv (T \circ J)(x)$. If for simplicity we assume that $(TJ)(x) \neq \pm\infty$ then $T : \mathsf{R}(X) \to \mathsf{R}(X)$.

Likewise define the mapping T_μ as

$$(T_\mu J)(x) = H(x, \mu(x), J) \ \ \forall x \in X$$

Our goal is to find $J^* \in \mathsf{R}(X)$ such that:

$$J^*(x) = \sup_{u \in U(x)} H(x, u, J^*) \ \ \forall x \in X,$$

which can be expressed compactly as

$$J^* = TJ^*,$$

Corresponding with J^* we wish to find a μ^* which fulfills:

$$H(x, \mu^*(x), J^*) \ = \sup_{u \in U(x)} H(x, u, J^*) \ \ \forall x \in X$$

$$\Rightarrow \mu^*(x) = \underset{u \in U(x)}{argsup}\, H(x, u, J^*) \ \ \forall x \in X$$

We can express the relationship between μ^* and J^* compactly as

$$T_{\mu^*} J^* \ = TJ^*.$$

For our problem we will define H as:

$$H(x, u, J) = r(x) + \beta \mathbb{E}\left[J(f(x, u, w))\right]$$

where the expectation is taken with respect to the random variable w.

A.1 Proof that T and T_μ are Contraction Mappings

A.1.1 Background Review

Before proving that T and T_μ are contraction mappings, we review some key concepts.

Define a metric space $(B(X), \rho_\infty)$ where $B(X)$ is the set of all continuous, bounded functions $f : X \to \mathbb{R}$ and ρ_∞ is the sup norm:

$$\rho_\infty(f, g) = \sup_{x \in X} \mid f(x) - g(x) \mid \quad \forall\, f, g \in B(X).$$

Define the operation $\leq$ on $B(X)$ as follows: $\forall f, g \in B(X),\ f \leq g \iff f(x) \leq g(x)\ \forall\, x \in X$.

Thus $(B(X), \rho_\infty)$ is a complete metric space. Let $T : B(X) \to B(X)$ be a mapping. Recall that T is a **contraction mapping** if $\exists \beta \in (0, 1]$ such that

$$\rho_\infty(Tf, Tg) \leq \beta \rho_\infty(f, g)\ \ \forall f, g \in B(X),$$

where we use the shorthand $Tf \equiv T(f)$.

Recall that **Blackwell's Sufficiency Conditions** provide a flexible and powerful set of conditions under which T is a contraction. These are:

- monotonicity: $\forall f, g \in B(X), f \leq g \to Tf \leq Tg$, and
- discounting: define $c : X \to \bar{c}$ for $\bar{c} \in \mathbb{R}^+$; then T discounts if

$$T(f + c) \leq Tf + \beta \bar{c}\ \ for\ any\ \beta \in (0, 1].$$

If T fulfills monotonicity and discounting, then T is a contraction mapping on $(B(X), \rho_\infty)$.

We care about Blackwell's Sufficiency Conditions because if some mapping: $T : B(X) \to B(X)$. can be shown to be a contration on a complete metric space (such as $(B(X), \rho_\infty)$), then we know that

- there is a **unique fixed point** $g^* \in B(X)$ *s.t.* $g^* = Tg^*$, and
- we can **construct a sequence** $\{g^k\}$ **which converges to** g^* as follows: let $g^k = T^k g^0$, where the notation T^k indicates that the mapping has been recursively applied to g^0. point can be constructed as the limit $\lim_{k\to\infty} g^k = g^*$, where $g^k = T^k g^0$, and g^0 is any point in $B(X)$.

We are now ready to show that T and T_μ are contraction mappings in $(B(X), \rho_\infty)$.

A.1.2 T and T_μ are Contraction Mappings

Consider the complete metric space $(B(X), \rho_\infty)$ where $B(X)$ is the set of all continuous, bounded functions $f : X \to \mathbb{R}$ and ρ_∞ is the sup norm.

We start with two lemmas:

Monotonicity of H:

Let H be defined as above and let $J, J^{'} \in B(X)$, where $J \leq J^{'}$. Observe that:

$$\begin{aligned} H(x,u,J) &= r(x) + \beta \mathbb{E}\left[J(f(x,u,w))\right] \ \forall\ x \in X, u \in U \\ &\leq r(x) + \beta \mathbb{E}\left[J^{'}(f(x,u,w))\right] \ \forall\ x \in X, u \in U \\ &= H(x,u,J') \ \forall\ x \in X, u \in U, u \in U \end{aligned}$$

Thus $J \leq J^{'} \to H(x,u,J) \leq H(x,u,J^{'}) \ \forall\ x \in X, u \in U$. $\square$

Discounting of H:

Let H be defined as above and let $J, c \in B(X)$, where $c : X \to \bar{c}$, and $\bar{c} \in \mathbb{R}^+$ is some constant. Observe that:

$$\begin{aligned} H(x, u, J + c) &= r(x) + \beta \mathbb{E}\left[J(f(x, u, w)) + c(x)\right] \ \forall \ x \in X, u \in U \\ &= r(x) + \beta \mathbb{E}\left[J(f(x, u, w))\right] + \beta c(x) \ \forall \ x \in X, u \in U \\ &= H(x, u, J) + \beta \bar{c} \ \forall \ x \in X, u \in U. \end{aligned}$$

Thus for all $J, c \in B(X)$ $s.t.$ $c : X \to \bar{c} \in \mathbb{R}$, we have that $H(x, u, J + c) = H(x, u, J) + \beta \bar{c} \ \forall \ x \in X, u \in U$. $\square$

Now consider T and T_μ as defined above. We can see that both are contraction mappings as follows:

Monotonicity:

Let $J, J^{'} \in B(X)$, where $J \leq J^{'}$. Observe that:

$$T_\mu J = H(x, \mu(x), J) \; \forall \; x \in X$$

$$\leq H(x, \mu(x), J) \; \forall \; x \in X$$

$$= T_\mu J^{'}$$

and

$$TJ = \sup_{u \in U(x)} H(x, u, J) \; \forall \; x \in X$$

$$\leq \sup_{u \in U(x)} H(x, u, J^{'}) \; \forall \; x \in X$$

$$= TJ^{'}$$

Thus $J \leq J^{'} \rightarrow T_\mu J \leq T_\mu J^{'}$ and $TJ \leq TJ^{'}$. $\square$

Discounting:

Let $J, c \in B(X)$, where $c : X \rightarrow \bar{c}$, and $\bar{c} \in \mathbb{R}^+$ is some constant. Observe that:

$$T_\mu(J+c) = H(x,\mu(x),J+c)\ \forall\ x \in X$$

$$= H(x,\mu(x),J) + \beta\bar{c}\ \forall\ x \in X$$

$$= T_\mu J + \beta\bar{c}$$

and

$$T(J+c) = \sup_{u\in U(x)} H(x,u,J+c)\ \forall\ x \in X$$

$$= \sup_{u\in U(x)} \{H(x,\mu(x),J) + \beta\bar{c}\}\ \forall\ x \in X$$

$$= \sup_{u\in U(x)} \{H(x,\mu(x),J)\} + \beta\bar{c}\ \forall\ x \in X$$

$$= TJ + \beta\bar{c}$$

Thus for all $J, c \in B(X)$ *s.t.* $c : X \to \bar{c} \in \mathbb{R}$, we have that $T_\mu(J+c) = T_\mu J + \beta\bar{c}$ and $T(J+c) = TJ + \beta\bar{c}$. $\square$

For a given policy μ we can define $T_\mu = H(x,\mu(x)$

By Blackwell's Sufficiency Conditions, we have that both T_μ and T are contraction mappings on the complete space $(B(X), \rho_\infty)$, and thus each have a unique fixed point which is attainable by an appropriately constructed sequence in $B(X)$.

A.2 Proof of Convergence of Policy Iteration

A.2.1 Policy Iteration Algorithm

Define the **Policy Iteration algorithm** as follows:

- Step (0) - **Initialization**: Choose an arbitrary initial policy $\mu^0 \in \mathcal{M}$
- Step (1) - **Policy Evaluation**: Given policy μ^k, find value function J_{μ^k} which is the fixed point of T_{μ^k}; that is:

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}$$

.

- Step (2) - **Policy Improvement**: Obtain new policy μ^{k+1} which solves $H(x, \mu^{k+1}(x), J_{\mu^k}) = \sup_{u \in U(x)} H(x, u, J_{\mu^k})$. That is, given J_{μ^k}, find the new policy μ^{k+1} consistent with

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}.$$

Note that the policy improvement step can be satisfied by constructing μ^{k+1} as the "greedy" policy with respect to $H(x, u, J_{\mu^k})$:

$$\mu^{k+1}(x) = \underset{u \in U(x)}{argsup}\, H(x, u, J_{\mu^k}) \; \forall \; x \in X.$$

Repeat the process at step 1 using the new policy μ^{k+1}.

Before proceeding we prove the following lemma:

Optimal Mapping Upper Bound (OMUB) Lemma:

Let μ be any continuous policy function and $J \in B(X)$ be any continuous value function such that the following relationship holds:

$$TJ \geq J.$$

We want to show that

$$T^n J \geq J \ \ \forall\, n \in \mathbb{N}$$

We proceed by standard induction:

- Base case: the hypothesis holds for $n = 1$ by assumption: $TJ \geq J$.
- General case assumption: assume the relationship holds for arbitrary $n = i$: $T^n J \geq J$.
- Induction step: We can now see that $n = i + 1$ holds as well:

$$T^i J \geq J \quad \text{by induction step}$$

$$\Rightarrow T(T^i J) \geq TJ \quad \text{by monotonicity of T}$$

$$\Rightarrow T^{i+1} J \geq TJ \geq J. \quad \text{by initial assumption}$$

Thus $T^n J \geq J \ \ \forall\, n \in \mathbb{N}.\square$

A.2.2 Policy Iteration Convergence

For an arbitrary k, the policy iteration step implies the following equivalences:

$$\begin{aligned} T_{\mu^{k+1}} J_{\mu^k} &= T J_{\mu^k} && \text{by the policy improvement step,} \\ &\geq T_{\mu^k} J_{\mu^k} && \text{by the optimality of } T, \\ &= J_{\mu^k} && \text{by the policy evaluation step,} \end{aligned}$$

or more compactly:

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k} \geq T_{\mu^k} J_{\mu^k} = J_{\mu^k}.$$

We get two useful inequalities out of this:

$$T J_{\mu^k} \geq J_{\mu^k} \text{ and} \tag{A.1}$$

$$T_{\mu^{k+1}} J_{\mu^k} \geq J_{\mu^k} \tag{A.2}$$

Using lemma (OMUB) and equation (A.1) immediately produces the following result:

$$T^n J_{\mu^k} \geq J_{\mu^k} \forall\ n,\ \ \forall\ k. \tag{A.3}$$

By (A.3) and the fact that $\lim_{n\to\infty} T^n J_0 = J^*$, where $J_0 \in B(X)$ is any initial function, we know that

$$J^* \geq J_{\mu^k} \forall\ k..$$

Using equation (A.2) we can prove an intermediate result. Observe that

$$T_{\mu^{k+1}} J_{\mu^k} \geq J_{\mu^k}$$

$$\Rightarrow T^2_{\mu^{k+1}} J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \quad \text{by monotonicity of T}$$

We can show, again by induction, that $T^n_{\mu^{k+1}} J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \forall\ n \geq 2$:

- The hypothesis holds for $n = 2$ by equation (A.2.2)
- Assume it holds for arbitrary $n = i : T^i_{\mu^{k+1}} J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k}$.
- We can now see that the steps $n = i + 1$ holds as well:

$$T^{i}_{\mu^{k+1}} J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \quad \text{by induction step,}$$

$$\Rightarrow T^{i+1}_{\mu^{k+1}} J_{\mu^k} \geq T^{2}_{\mu^{k+1}} J_{\mu^k} \quad \text{by monotonicity of } T_{\mu^{k+1}},$$

$$\geq T_{\mu^{k+1}} J_{\mu^k},$$

where the final inequality holds by equation (A.2.2). Thus $T^{n}_{\mu^{k+1}} J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \forall\ n \geq 2$ by induction, and since $\lim_{n\to\infty} T^{n}_{\mu^{k+1}} J_0 = J_{\mu^{k+1}}$, where J_0 is any initial function in B(X)$, we know that

$$J_{\mu^{k+1}} \geq T_{\mu^{k+1}} J_{\mu^k}.$$

We can combine this with equation (A.2.2) to get the following key inequality:

$$J_{\mu^{k+1}} \geq T J_{\mu^k} \geq J_{\mu^k}.$$

For the sake of the following proof, restate this as:

$$J_{\mu^k} \geq T J_{\mu^{k-1}} \geq J_{\mu^{k-1}}.$$

We will use induction to show that $J_{\mu^n} \geq T^n J_{\mu^0}\ \forall\ n$:

- The hypothesis holds for $n = 1$ by equation (A.2.2)
- Assume it holds for arbitrary $n = i : J_{\mu^i} \geq T^i J_{\mu^0}$.
- We can now see that the steps $n = i + 1$ holds as well:

$$T J_{\mu^i} \geq T^{i+1} J_{\mu^0} \text{ by inductive step and monotonicity of T}$$

$$\Rightarrow J_{\mu^{i+1}} \geq T J_{\mu^i} \geq T^{i+1} J_{\mu^0}.$$

Thus by induction

$$J_{\mu^k} \geq T^k J_{\mu^0} \; \forall \; k.$$

Together equations (A.2.2) and (A.2.2) imply

$$J^* \geq J_{\mu^k} \geq T^k J_{\mu^0} \; \forall \; k.$$

By the fact that T is a contraction mapping on $(B(X), \rho_\infty)$, we know that for any $J_0 \in B(X)$,

$$\lim_{k \to \infty} \rho_\infty (T^k J_0 - J^*) = 0$$

$$\Rightarrow \lim_{k \to \infty} \rho_\infty (T^k J_{\mu^0} - J^*) = 0$$

This limit along with inequality (A.2.2) implies that

$$\lim_{k \to \infty} \rho_\infty (J_{\mu^k} - J^*) = 0,$$

and thus the policy iteration algorithm is squeezed to converged to the optimal policy by the value iteration iterates.

Note that the step (A.2.2) is a proof for the convergence of value iteration, which is simply attained by the repeated application of T to an arbitrary intial value function.

Interestingly enough, this final step in the proof gives us analytical intuition for the reduced convergence times experienced by practitioners using policy iteration: for each iteration step k, the policy iteration algorithm produces a value function J_{μ^k} which in the *worst case* is as far from the optimal value function J^* as the equivalent k^{th}-iterate of value iteration.

A.3 Proof of Convergence of Optimistic Policy Iteration

A.3.1 Optimistic Policy Iteration Algorithm

Define the **Optimistic Policy Iteration (OPI) algorithm** as follows:

Algorithm I:

- Step 0 - **Initialization**: Choose an arbitrary initial value function J_0 such that $TJ_0 \geq J_0$.[1]. Choose a sequence of integers $\{m_k\}_{k=0}^{\infty}$, where $m_k \in \mathbb{N} \; \forall \; k \in \mathbb{N}$.

- Step 1 - **Policy Improvement**: Obtain policy μ^k which solves $H(x, \mu^k(x), J_k) = \underset{u \in U(x)}{sup} \; H(x, u, J_k)$. That is, given J_k, find the policy μ^k consistent with

$$T_{\mu^k} J_k = TJ_k.$$

Note that the policy improvement step can be satisfied by constructing μ^k as the "greedy" policy with respect to $H(x, u, J_k)$:

$$\mu^k(x) = \underset{u \in U(x)}{argsup} \; H(x, u, J_k) \; \forall \; x \in X.$$

- Step 2 - **Optimistic Policy Evaluation**: Given value function J^k and policy μ_k, execute the optimistic value function update to obtain J_{k+1}, which results from the application of mapping T_{μ^k} to J^k m_k times. That is:

$$J_{k+1} = T_{\mu^k}^{m_k} J_k.$$

Repeat the process at step 1 using the new value function J_k.

[1]Note that such an initial function can always be constructed from any arbitrary function $\widetilde{J}$ as follows. Choose a constant $c = \underset{x \in X}{max} \; \rho_\infty(T\widetilde{J} - \widetilde{J})$, then define $J_0 = \widetilde{J} + c$.

Note that an **alternative** statement of this process can be outlined in a slightly different order. If both μ^0 and J^0 are choosen, we can proceed as follows:

Algorithm II:

- Step 0 - **Initialization**: Choose an arbitrary initial value and policy function, J_0 and μ^0, and a sequence of integers $\{m_k\}_{k=0}^{\infty}$, where $m_k \in \mathbb{N} \; \forall \; k \in \mathbb{N}$.

- Step 1 - **Optimistic Policy Evaluation**: Given value function J^{k-1} and policy μ_{k-1}, find the optimistic value associated with the policy function J_k which results from the application of mapping $T_{\mu^{k-1}}$ to J^{k-1} m_k times. That is:

$$J_k = T_{\mu^{k-1}}^{m_{k-1}} J_{k-1}$$

.

- Step 2 - **Policy Improvement**: Obtain new policy μ^k which solves $H(x, \mu^k(x), J_k) = \sup_{u \in U(x)} H(x, u, J_k)$. That is, given J_k, find the policy μ^k consistent with

$$T_{\mu^k} J_k = T J_k.$$

Note that the policy improvement step can be satisfied by constructing μ^k as the "greedy" policy with respect to $H(x, u, J_k)$:

$$\mu^k(x) = \underset{u \in U(x)}{argsup} \; H(x, u, J_k) \; \forall \; x \in X.$$

We will first prove convergence for the more-traditional Optimistic Policy Iteration Algorithm I. The proof will follow a very similar pattern to that of the Policy Iteration algorithm.

As with policy iteration, we want to prove the following inequality holds, which will allow us to "squeeze" the optimistic policy iterates to the optimal function:

$$J^* \geq J_k \geq T^k J_0 \; \forall \; k \in \mathbb{N}.$$

That is, we want to show that the equivalent value function iteration at step k is always at best as far from the optimal value function as the equivalent value function from optimistic policy iteration.

Before proceeding we state the following lemmas:

Multistep Policy Mapping I Lemma (MPM I):

Let μ be any continuous policy function and $J \in B(X)$ be any continuous value function such that the following relationship holds: $T_\mu J \geq J$.

We want to show that for any $m \in \mathbb{N}$,

$$T_\mu^{m+1} J \geq T_\mu^m J.$$

For consistency we define T_μ^0 as simply no application of the mapping T_μ. As always we proceed by induction.

- Base case: we consider two steps for completeness:
  - $m = 0$: This holds by the lemma assumption: $T_\mu^1 J \geq T_\mu^0 J = T_\mu J \geq J$
  - $m = 1$: This holds by the lemma assumption and monotonicity of T_μ: $T_\mu^2 J \geq T_\mu J$
- General case assumption: Assume the general case holds for $m = i$:

$$T_\mu^{i+1} J \geq T_\mu^i J$$

- Induction step: we can see now that the general case for $m = i$ implies that the relationship continues to hold for $m = i + 1$:

$$T_\mu^{i+1} J \geq T_\mu^i J \qquad \text{by the general case assumption,}$$

$$\Rightarrow T_\mu(T_\mu^{i+1} J) \geq T_\mu(T_\mu^i J) \qquad \text{by applying } T_\mu \text{ mapping to both sides;}$$

$$\text{inequality holds by monotonicity of } T_\mu,$$

$$\Rightarrow T_\mu^{i+2} J \geq T_\mu^{i+1} J \qquad \text{and achieve the induction result.}$$

Thus by induction we know that

$$T_\mu J \geq J \Rightarrow T_\mu^{m+1} J \geq T_\mu^m J \; \forall \; m \in \mathbb{N}. \square$$

Multistep Policy Mapping II Lemma (MPM II):

Let μ be any continuous policy function and $J \in B(X)$ be any continuous value function such that the following relationship holds: $T_\mu J \geq J$.

We want to show that for any $m \in \mathbb{N}$,

$$T_\mu^m J \geq T_\mu J.$$

For consistency we define T_μ^0 as simply no application of the mapping T_μ. We again proceed by induction on m:

- Base cases:
  - $m = 1$: This holds by definition: $T_\mu J = T_\mu J$.
  - $m = 2$: This holds by the assumption $T_\mu J \geq J$ and monotonicity of T_μ: $T_\mu^2 J \geq T_\mu J$.
- General case: Assume the general case holds for $m = i$:

$$T_\mu^i J \geq T_\mu J$$

- Induction step: the general case for $m = i$ implies that the relationship holds for $m = i + 1$:

$$T_\mu^i J \geq J$$

$$\Rightarrow T(T_\mu^i J) \geq T_\mu J \quad \text{by general case assumption, monotonicity of } T_\mu$$

$$\Leftrightarrow T_\mu^{i+1} J \geq T_\mu J.$$

Thus by induction we know that

$$T_\mu J \geq J \Rightarrow T_\mu^m J \geq T_\mu J \ \ \forall\, m \in \mathbb{N}.\square$$

Multistep Policy Mapping III Lemma (MPM III):

Consider a sequence of policy functions $\{\mu^k\}$ and value functions $\{J_k\}$ generated by the Optimistic Policy Iteration algorithm using sequence $\{m_k\}_{k=0}^{\infty}$. Then the following relationship must hold for all $m \in \mathbb{N}$:

$$T_{\mu^k}^{1+m} J_k \geq T_{\mu^k}^m J_k.$$

Once this is demonstrated for all m, it is clear that it will hold for the specific case $T_{\mu^k}^{1+m_k} J_k \geq T_{\mu^k}^{m_k} J_k$, where m is replaced with the values $m_k \in \{m_k\}$ defined above.

We will prove (A.3.1) by a nested induction argument.

First consider induction on m. We first show that, for $k = 0$, the relationship $T_{\mu^0}^{m+1} J_0 \geq T_{\mu^0}^m J_0$ holds for all $m > 0, m \in \mathbb{N}$:

- Initial steps:
  - $m = 0$ and we define T^0 as no transformation; this holds by the initial assumption and the policy improvement step: $T_{\mu^0} J_0 = T J_0 \geq J_0$

– $m = 1$ holds by the $m = 0$ case above and the monotonicity of T_{μ^0}: $T^2_{\mu^0} J_0 \geq T_{\mu^0} J_0$

- Assume the relationship holds for general $m = i$: $T^{i+1}_{\mu^0} J_0 \geq T^i_{\mu^0} J_0$
- Induction step: we can now immediately see that this holds for $m = i + 1$ as well:

$$T^{i+1}_{\mu^0} J_0 \geq T^i_{\mu^0} J_0$$

$$\Rightarrow T_{\mu^0}(T^{i+1}_{\mu^0} J_0) \geq T_{\mu^0}(T^i_{\mu^0} J_0) \quad \text{by assumption and monotonicity of } T_{\mu^0}$$

$$\Leftrightarrow T^{i+2}_{\mu^0} J_0 \geq T^{i+1}_{\mu^0} J_0$$

$$\Rightarrow T^{m+1}_{\mu^0} J_0 \geq T^m_{\mu^0} J_0, \ \forall \ m \in \mathbb{N}. \square$$

Thus by induction it must be the case that for all $m > 0, m \in \mathbb{N}, T^{m+1}_{\mu^0} J_0 \geq T^m_{\mu^0} J_0$, so the specific case of $m = m_0$ holds as well.

Note that this conveniently defines for us the first step in an induction proof for demonstrating that, **for all k**, $T^{1+m}_{\mu^k} J_k \geq T^m_{\mu^k} J_k$:

- Initial step, $k = 0$: $T^{1+m}_{\mu^0} J_0 \geq T^m_{\mu^0} J_0$ was proved above $\forall \ m$.
- We assume the relationship holds for general $k = i$:

$$T^{1+m}_{\mu^i} J_i \geq T^m_{\mu^i} J_i \ \ \forall \ m$$

- Inductive step: the general case above implies that relationship continues to hold for $k = i + 1$. We can see that:

$$
\begin{aligned}
T_{\mu^{i+1}} J_{i+1} &= T J_{i+1} && \text{by OPI Policy Improvement step} \\
&\geq T_{\mu^i} J_{i+1} && \text{by definition of } \mu^i \text{ in OPI} \\
&= T_{\mu^i}^{1+m_i} J_i && \text{by OPI policy evaluation step and monotonicity of } T_{\mu^i} \\
&\geq T_{\mu^i}^{m_i} J_i && \text{by general case assumption for } k = i, m = m_i \\
&= J_{i+1} && \text{by OPI policy evaluation step}
\end{aligned}
$$

$$\Rightarrow T_{\mu^{i+1}} J_{i+1} \geq J_{i+1}$$

$$\Rightarrow T_{\mu^{i+1}}^{m+1} J_{i+1} \geq T_{\mu^{i+1}}^{m} J_{i+1} \forall\ m \in \mathbb{N} \quad \text{by the (MPM II) Lemma}$$

$$\Rightarrow T_{\mu^k}^{m+1} J_k \geq T_{\mu^k}^{m} J_k\ \forall\ k \in \mathbb{N} \ \text{ and } \ \forall\ m \in \mathbb{N}. \square$$

Since this holds for all m, it is clear that it also holds for the sequence $\{m_k\}$. Note that the general-m result is imporant in its own right.

We use this result to immediately write another lemma and solidify an important inequality:

Multistep Policy Mapping IV Lemma (MPM IV):

Consider a sequence of policy functions $\{\mu^k\}$ and value functions $\{J_k\}$ generated by the Optimistic Policy Iteration algorithm. Then the following relationship must hold for all $k \in \mathbb{N}$:

$$T_{\mu^k} J_k \geq J_k.$$

This can be show directly as follows. For all k, the following inequalities hold:

$$\begin{aligned}
T_{\mu^k} J_k &= T J_k \quad \text{by OPI policy improvement step} \\
&\geq T_{\mu^{k-1}} J_k \quad \text{by definition of } T_{\mu^{k-1}} \text{ in OPI} \\
&= T_{\mu^{k-1}}^{1+m_{k-1}} J_{k-1} \quad \text{by OPI policy evaluation } (J_k = T_{\mu^{k-1}}^{m_{k-1}} J_{k-1}), \text{ monotonicity of } T_{\mu^{k-1}} \\
&\geq T_{\mu^{k-1}}^{m_{k-1}} J_{k-1} \quad \text{by Lemma (MPM III)} \\
&= J_k \quad \text{by OPI policy evaluation step}
\end{aligned}$$

$\Rightarrow T_{\mu^k} J_k \geq J_k \ \ \forall k \in \mathbb{N}. \square$

Multistep Policy Mapping V Lemma (MPM V):

Consider a sequence of policy functions $\{\mu^k\}$ and value functions $\{J_k\}$ generated by the Optimistic Policy Iteration algorithm. Then the following relationship must hold for all $k \in \mathbb{N}$:

$$J_{k+1} \geq T J_k.$$

This can be show directly as follows:

$$J_{k+1} = T_{\mu^k}^{m_k} J_k \quad \text{by OPI policy evaluation step}$$

$$\geq T_{\mu^k} J_k \quad \text{by (MPM IV) and (MPM II)}$$

$$= TJ_k \quad \text{by OPI policy improvement step.}$$

$$\Rightarrow J_{k+1} \geq TJ_k \ \ \forall k \in \mathbb{N}. \square$$

Multistep Policy Mapping VI Lemma (MPM VI):

Consider a sequence of policy functions $\{\mu^k\}$ and value functions $\{J_k\}$ generated by the Optimistic Policy Iteration algorithm. Then the following relationship must hold for all $k \in \mathbb{N}$:

$$J_k \geq T^k J_0.$$

This can be show by induction as follows:

- Base case $k = 1$: $J_1 \geq TJ_0$ holds by lemma (MPM-V).
- General case assumed for $k = n$:

$$J_n \geq T^n J_0.$$

- Inductive step: the general case implies the $k = n + 1$ case:

$$J_n \geq T^n J_0$$

$$\Rightarrow TJ_n \geq T^{n+1} J_0 \quad \text{by general case and monotonicity of } T$$

$$\Rightarrow J_{n+1} \geq TJ_n \geq T^{n+1} J_0 \quad \text{by lemma (MPM-V).}$$

Thus by induction,

$$J_k \geq T^k J_0 \ \ \forall k \in \mathbb{N}.\square$$

Multistep Policy Mapping VII Lemma (MPM VII):

Consider a sequence of policy functions $\{\mu^k\}$ and value functions $\{J_k\}$ generated by the Optimistic Policy Iteration algorithm. Then the following relationship must hold for all $k \in \mathbb{N}$:

$$TJ_k \geq J_k.$$

This can be show directly as follows. For all k, the following inequalities hold:

$T_{\mu^k} J_k = TJ_k$ by OPI policy improvement step

$\geq T_{\mu^{k-1}} J_k$ by definition of $T_{\mu^{k-1}}$ in OPI

$= T_{\mu^{k-1}}^{1+m_{k-1}} J_{k-1}$ by OPI policy evaluation ($J_k = T_{\mu^{k-1}}^{m_{k-1}} J_{k-1}$), monotonicity of $T_{\mu^{k-1}}$

$\geq T_{\mu^{k-1}}^{m_{k-1}} J_{k-1}$ by Lemma (MPM III)

$= J_k$ by OPI policy evaluation step

$\Rightarrow TJ_k \geq J_k \ \ \forall k \in \mathbb{N}.\square$

Multistep Inequality Property (MIP) and convergence of OPI:

We are now in a position to prove the convergence of Optimistic Policy Iteration. Lemmas (MPM VII) and (OMUB) immediately give us $T^n J_k \geq J_k \ \ \forall\, n, k \in \mathbb{N}$. The fact that $\lim_{n\to\infty} T^n J_0 = J^*$, where $J_0 \in B(X)$ is any initial function produces the following useful

result:

$$J^* \geq J_k \forall\ k,$$

This together with lemma (MPM VI) produces the key inequality:

$$J^* \geq J_k \geq T^k J_0 \ \ \forall k \in \mathbb{N}.$$

That is, sequence of value functions produced by the optimistic policy iteration algorithm, $\{J_k\}_{k=0}^{\infty}$, is bounded above by the optimal value function, and below by the sequence of value functions produced by applying the optimal mapping T to J_0 k times.

By the fact that T is a contraction mapping on $(B(X), \rho_\infty)$, we know that for any $J_0 \in B(X)$,

$$\lim_{k\to\infty} \rho_\infty(T^k J_0 - J^*) = 0,$$

This limit along with inequality (A.3.1) implies that

$$\lim_{k\to\infty} \rho_\infty(J_k - J^*) = 0.$$

Thus, just as with the policy iteration algorithm, the optimistic policy iteration algorithm is squeezed to converged to the optimal policy by the value iteration iterates.

Convergence of OPI, Alternative Formulation:

The alternative formulation of Optimistic Policy Iteration, "Algorithm II," in which both initial policy and value functions are chosen and the policy evaluation step is executed first, can be shown to converge to the optimal policy under the same conditions as "Algorithm I," after one initial issue is addressed. The convergence of OPI, algorithm I, shown above, relies on the fact that the intial function starts "below" the optimal function. That is,

$$TJ_0 \geq J_0.$$

For the OPI algorithm II to converge, all we need to demonstrate is that the chosen μ_0 and J_0 produce a J_1 in the first policy evaluation step such that

$$TJ_1 \geq J_1;$$

in which case the rest of the proofs above follow exactly as before, with J_1 taking the place of J_0 (and indices all shifted forward by 1 as needed). We will prove directly that $TJ_1 \geq J_1$ under particular conditions for μ_0 and J_0.

Let μ_0 and J_0 be defined such that $T_{\mu^0}J_0 \geq J_0$. Then it is clear that

$TJ_1 = T(T_{\mu^0}^{m_0}J_0)$ by the first policy evaluation step and monotonicity of T

$\geq T(T_{\mu^0}^{m_0-1}J_0)$ by assumption $T_{\mu^0}J_0 \geq J_0$, lemma (MPM I), and monotonicity of T

$\geq T_{\mu^0}(T_{\mu^0}^{m_0-1}J_0)$ by the optimality of T

$= T_{\mu^0}^{m_0}J_0$

$= J_1.\square$

Thus $T_{\mu^0}J_0 \geq J_0$ and the first policy evaluation step imply that

$$TJ_1 \geq J_1,$$

and the convergence of OPI Algorithm II follows from the proof for convergence of OPI Algorithm I.

Appendix B: Allen and Carroll's Individual Learning Algorithm

Allen and Carroll's "individual learning" (IL) algorithm is an excellent starting point for framing regret learning (RL). Consider the agent problem presented in expression (C.6), restated here for convenience:

$$\max_{\{c_t\}_{t=0}^{\infty}} \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right] \tag{B.1}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1} \text{ the law of motion,}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \textit{ given.}$$

Allen and Carroll begin by discretizing the state space into N representative points. Their original parameterization choose a discrete space:[1]

$$M_{discrete} \approx \{1, 2, 3\}. \tag{B.2}$$

Individual-learning agents were then assigned assigned two learning parameters – N_T, the number of time periods from which to learn, and N_J, the number of experiences of

[1]The approximation is due to the fact that each agent i was started with initial *savings* of $s_0 \equiv R(m_{-1} - c_{-1}) \in \{0, 1, 2\}$ which in the simulation was then immediately turned into cash-on-hand by the addition of the initial income shock: $m_0 = y_0$ for each agent i. This was employed due to technical constraints on the original simulation.

length N_T to generate per rule – and a set of 400 parameterized consumption functions c^θ where $\theta \in \Theta$; $\#\{\Theta\} = 400$. The particular parameterized form of the consumption function is described in the main paper but is not important for our purposes here.

For a given policy rule parameter θ, each agent estimates a "single run" value $w_j^\theta(m)$ as

$$w_j^\theta(m_0) = \sum_{t=0}^{N_T} \beta^t u(c^\theta(m_t)), \tag{B.3}$$

where the vector $\vec{m}_i = [m_0, m_1, ..., m_{N_T}]$ for each agent would be generated by a draw of income shocks for each agent $\vec{y}_i = [y_0, y_1, ..., y_{N_T}]$, their initial value of m_0 as described above, and the law of motion indicated in their problem (2.3). Each agent would generate N_J sets of experiences; that is, N_J single run values $w_j^\theta(m_0)$. Importantly, each "single run" must be experienced by the agent as a sort of "groundhog day" – each experience began with the agent restating at their initial savings level, which provided for a consistent estimator.[2] These would together be used to construct$\bar{w}_i^\theta(m_0)$:

$$\bar{w}_i^\theta(m_0) = \frac{1}{N_J} \sum_{j=1}^{N_J} w_m^\theta(m_0). \tag{B.4}$$

Jointly sending $(N_T, N_J) \to (\infty, \infty)$ implies that $\bar{w}_i^\theta(m_0)$ is a consistent estimator of

$$E[V^\theta \mid X_0] = \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(C^\theta(X_t)) \right]. \tag{B.5}$$

The question becomes the following: what values of N_T and N_J will allow a large

[2]Allen and Carroll's learning algorithm is in fact almost exactly identical to so-called "first-visit" Monte Carlo learning in the Reinforcement Learning literature in computer science. An excellent early paper which oulines the basic theoretical properties of first-visit estimators is Singh and Sutton (1996).

fraction of consumers to find a rule within some utility-distance of the true optimal rule? An excerpt of Allen and Carroll's initial results are displayed in table (B). 100 consumers were instantiated and execute a grid search of the Θ parameter space, implementing the estimation in equations (B.3) and (B.4). A "sacrifice value," which indicates the one-time payment required for an agent currently using the optimal rule c^* to be convinced to switch to the approximate rule c^θ indefinitely, can be calculated for each policy θ and is stated in terms of a fraction of annual income. The "success rate" in Table (B) refers to the fraction of the 100 consumers which choose a rule with a sacrifice value ≤ 0.05. A higher "success rate" for a given (N_T, N_J) pair means that agents are finding the near-optimal rule more consistently.

The positive result is that high enough (N_T, N_J) means that $> 90\%$ of the time, individual agents arrive at a near-optimal consumption policy. The negative result is that this takes 4 million periods. If we lower the requirement to $> 50\%$ of agents obtaining a near-optimal policy, we still need at least 200,000 periods to obtain this.[3] Much of this time is spent on the grid search – explicitly exploring rules which are of low value. However even if agents were particularly good or lucky in exploring the space and could find a near-optimal rule in say, 5 rule-trials, this still implies that a $> 50\%$ success rate requires $50 \times 10 \times 5 = 2,500$ periods – still ˜96 years even if the periods are biweekly versus annual.

Palmer (2015) demonstrates a simple extension to the agent problem which greatly reduces learning time, by roughly an order of magnitude. This improves the situation somewhat; the addition of a simple form of social learning in that paper helps bridge the gap further. Mechanically, however, agents in the model are still restricted to whatever initial set of rules, Θ, the researcher provides the agents to explore. While a simple exploration options might involve random search inside some parameter radius, or using some sampling distribution centered around the current parameter location (for example, a multivariate normal), this quickly begins to add parameters which do not have a simple or intuitive interpretation.For example, one might ask what the sampling radius should be, or what

[3]Even at a biweekly frequency, this implies roughly 8,000 years in agent time.

the variance on a sampling distribution would be. An economically intutive answer might look to normalize the radius in each parameter direction to be "one in utility terms" – however this would now require agents to either estimate the quantity themselves, or have the researcher solve for this quantity properly and hand it to the agent. This may raise even more quetions – if the agents have access to the ability to solve for this information, shouldn't they simply use that directly vs employing the learning scheme? Questions such as these motivated a search for a more intuitively appealing way to generate candidate consumption functions using knowledge already obtained by the agent. Of course, if the agent is to use his/her own experience, that is, the value function estimated from experience, it must inform the agent about a much greater selection of the state space. This quickly led to the regret-based learning which is the primary contribution of this paper.

Excerpt of Original Allen and Carroll Results

$m_0 \approx 2$					
		$N_J = 1$	$N_J = 10$	$N_J = 50$	$N_J = 200$
$N_T = 10$	Mean sacrifice:	0.269	0.122	0.100	0.102
	Success rate:	0.09	0.23	0.29	0.24
	Total periods:	4,000	40,000	200,000	800,000
$N_T = 20$	Mean sacrifice:	0.226	0.079	0.053	0.047
	Success rate:	0.18	0.45	0.62	0.68
	Total periods:	8,000	80,000	400,000	1.60E+06
$N_T = 50$	Mean sacrifice:	0.187	0.058	0.036	0.024
	Success rate:	0.26	0.58	0.76	0.91
	Total periods:	20,000	200,000	1.00E+06	4.00E+06

Appendix C: Extended Household Problem

The learning-to-optimize behavior I discuss in the main text is explicitly applied to a stationary, infinite-horizon dynamic optimization problem. In this appendix I describe a related finite-horizon problem. With minor modifications, this finite horizon problem may be transformed into an infinite-horizon problem which will serve as the learning target. Both problems are set up in the following section to anticipate estimation efforts which adopt elements of both problems.

C.0.2 Finite Horizon Consumer Problem

Consider the following finite-horizon consumption-under-uncertainty problem. At time $T+1$, the consumer dies with certainty. The problem is to allocate consumption appropriately from $t = 0$ to $t = T$. The full problem can be described as follows, with the notation closely follows that of Carroll (2012c). This setup can describe a wide range of consumer circumstances, including retirement and fixed income over final years of life, by properly defining the members of Γ_t, ψ_t, and ξ_t discussed below:

$$\max_{\{\mathbf{c}_t\}_{t=0}^{T}} \mathbb{E}_0 \left[\sum_{t=0}^{T} \hat{\beta}^t \not{\mathsf{D}}_t u(\mathbf{c}_t) \right] \tag{C.1}$$

$$s.t.$$

$$\mathbf{a}_t = \mathbf{m_t} - \mathbf{c_t}$$

$$\mathbf{b}_{t+1} = \mathbf{a_t}\mathsf{R}$$

$$\mathbf{p}_{t+1} = \mathbf{p_t}\Gamma_{t+1}\psi_{t+1} = \mathbf{p_t}\widetilde{\Gamma}_{t+1}$$

$$\mathbf{m}_{t+1} = \mathbf{b}_{t+1} + \mathbf{y}_{t+1}$$

$$\mathbf{y}_{t+1} = \mathbf{p}_{t+1}\xi_{t+1}$$

$$\mathbf{c_t} \geq 0$$

$$\mathbf{m_t} \geq -\bar{\mathbf{q}}_t$$

$$\mathbf{m}_0 \; given$$

where

- $\hat{\beta}$ is the discount rate,
- $\not{\mathsf{D}}_t$ is the independently distributed probability of staying alive through period t, with $\not{\mathsf{D}}_{T+1} = 0.0$
- $\mathbf{a}_t$ is end-of-period assets,
- $\mathbf{m}_t$ is beginning-of-period total market resources ("cash on hand"), $\mathbf{m} \in M \subset \mathbb{R}$,
- $\mathbf{c}_t$ is consumption in period t,
- R is a constant return factor on assets, $\mathsf{R} = (1 + r)$,
- $\mathbf{y}_t$ is income in period t,

- $\mathbf{p}_t$ is permanent non-asset income,
- Γ_t is deterministic permanent income growth factor,
- $\widetilde{\Gamma}_t$ is a combination of Γ_t and ψ_t for notational convenience,
- $-\bar{\mathbf{q}}_t$ is a borrowing constraint, $-\bar{\mathbf{q}}_t \in [0, \infty) \ \forall_t$,
- ψ_t is a mean-1 iid permanent shock to income, and
- ξ_t is a mean-1 iid transitory shock income, composed as

$$\xi_t = \begin{cases} 0 & with\ prob\ \wp_t > 0 \\ \frac{\phi_t}{\not\wp_t} & with\ prob\ \ \not\wp_t \equiv (1 - \wp_t) \end{cases}, \text{where}$$

- $\wp_t$ is a small probability that income will be zero
- ϕ_t is a mean-1 iid transitory shock to income

The utility function $u(.)$ if the objective function (C.1) is of the Constant Relative Risk Aversion (CRRA) form with risk-aversion parameter ρ:

$$u(c) = \frac{c^{(1-\rho)}}{1-\rho}.$$

The consumer dies with certainty at period T but may die with probability D_t in any other period t, where we assume that the arrival of death is independent each period. This implies that the probability of being alive in a period s can be calculated:

$$\not\mathsf{D}_s = \Pi_{j=0}^{s}(1 - \mathsf{D}_j)$$

or when $\mathsf{D}_j = \mathsf{D} \ \forall_j$, we can restate the expression in simpler terms:

$$\not{\mathsf{D}} \equiv (1 - \mathsf{D})$$

$$\rightarrow \not{\mathsf{D}}_s = (1 - \mathsf{D})^s \ \ \forall_s.$$

When death occurs, all subsequent period utility functions are 0. Note that the finite-horizon problem could be equivalently re-written with an infinite horizon but a utility function defined as

$$u(c_t) = \begin{cases} \frac{c^{(1-\rho)}}{1-\rho} & for\ t \leq T \\ 0 & for\ t > T. \end{cases}$$

This also indicates why $\not{\mathsf{D}}_s$ shows up in a clean form in the objective function: the expectation each period can be decomposed into the convex combination of expectations under either staying alive or dying in the following period. However the summation of payoffs following death is 0. This term falls out of the expression leaving only $\not{\mathsf{D}}_s$ in the objective.

As in Carroll (2012b), this problem can be normalized by permanent income $\mathbf{p}_t$ to produce a simplified version of the full problem with a reduced number of state variables. The bold symbols used above indicate non-normalized variables, while the regular non-bold symbols used below indicate variables normalized by permanent income. The normalized problem can be written in Bellman form:

$$v_t(m_t) = \max_{c_t} u(c_t) + \hat{\beta}\, \not{\mathsf{D}}_{t+1}\mathbb{E}_t\left[\widetilde{\Gamma}_{t+1}^{1-\rho} v_{t+1}(m_{t+1})\right] \tag{C.2}$$

$$s.t.$$

$$a_t = m_t - c_t$$

$$b_{t+1} = \left(\frac{\mathsf{R}}{\widetilde{\Gamma}_{t+1}}\right) a_t = \mathcal{R}_{t+1} a_t$$

$$m_{t+1} = b_{t+1} + \xi_{t+1}$$

$$c_t \geq 0$$

$$m_t \geq -\bar{q}$$

$$m_0 \; given,$$

or simplified further:

$$v_t(m_t) = \max_{c_t} u(c_t) + \hat{\beta}\, \not{\mathsf{D}}_{t+1}\mathbb{E}_t\left[\widetilde{\Gamma}_{t+1}^{1-\rho} v_{t+1}(m_{t+1})\right] \tag{C.3}$$

$$s.t.$$

$$m_{t+1} = \mathcal{R}_{t+1}(m_t - c_t) + \xi_{t+1}$$

$$c_t \geq 0$$

$$m_t \geq -\bar{q}$$

$$m_0 \; given.$$

C.0.3 Infinite Horizon Consumer Problem

Extending the above to a stationary infinite-horizon problem is straightforward. Intuitively, we must remove the elements which are not stationary. One simple way to obtain this is by setting the following:

- $\mathsf{D}_t = \mathsf{D} \ \forall_t$
- $\sigma_{\psi,t} = 0 \ \forall_t$ (or equivalently, $\psi_t = 1 \ \forall_t$)
- $\Gamma_t = 1 \ \forall_t$
- $\bar{q}_t = \bar{q} = 0 \ \forall_t$.

The income process is now stationary with only iid transitory shocks each period.[1] The probability of death now follows a Poisson process. We state the consumers problem in Bellman form as the following:

$$v(m_t) = \max_{c_t} \ u(c_t) + \cancel{\mathsf{D}}\hat{\beta}\mathbb{E}_t \left[v(m_{t+1})\right] \tag{C.4}$$

$$s.t.$$

$$a_t = m_t - c_t$$

$$b_{t+1} = \mathsf{R}a_t$$

$$m_{t+1} = b_{t+1} + y_{t+1}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \ given,$$

[1] Stationarity can be achieved with less restrictive assumptions; these are used to simplify exposition. Observe that under these conditions $y_t \equiv \xi_t$; we will use y_t for the duration of the paper. The conclusion discusses future work which retains the permanent shocks to income and $\Gamma_t = \Gamma \ \forall_t$ so the learning model may be matched against empirical data.

or simplified further:

$$v(m_t) = \max_{c_t} u(c_t) + \beta \mathbb{E}_t \left[v(m_{t+1}) \right] \tag{C.5}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \; given,$$

where the discount factor absorbs the probability of remaining alive $\beta \equiv \not{D}\hat{\beta}$. Note that we can restate the summation form of this simplified problem as:

$$\max_{\{c_t\}_{t=0}^{\infty}} \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right] \tag{C.6}$$

$$s.t.$$

$$m_{t+1} = \mathsf{R}(m_t - c_t) + y_{t+1} \;\text{ the law of motion,}$$

$$c_t \geq 0$$

$$m_t \geq 0$$

$$m_0 \; given.$$

.

Appendix D: Figures for Social Learning

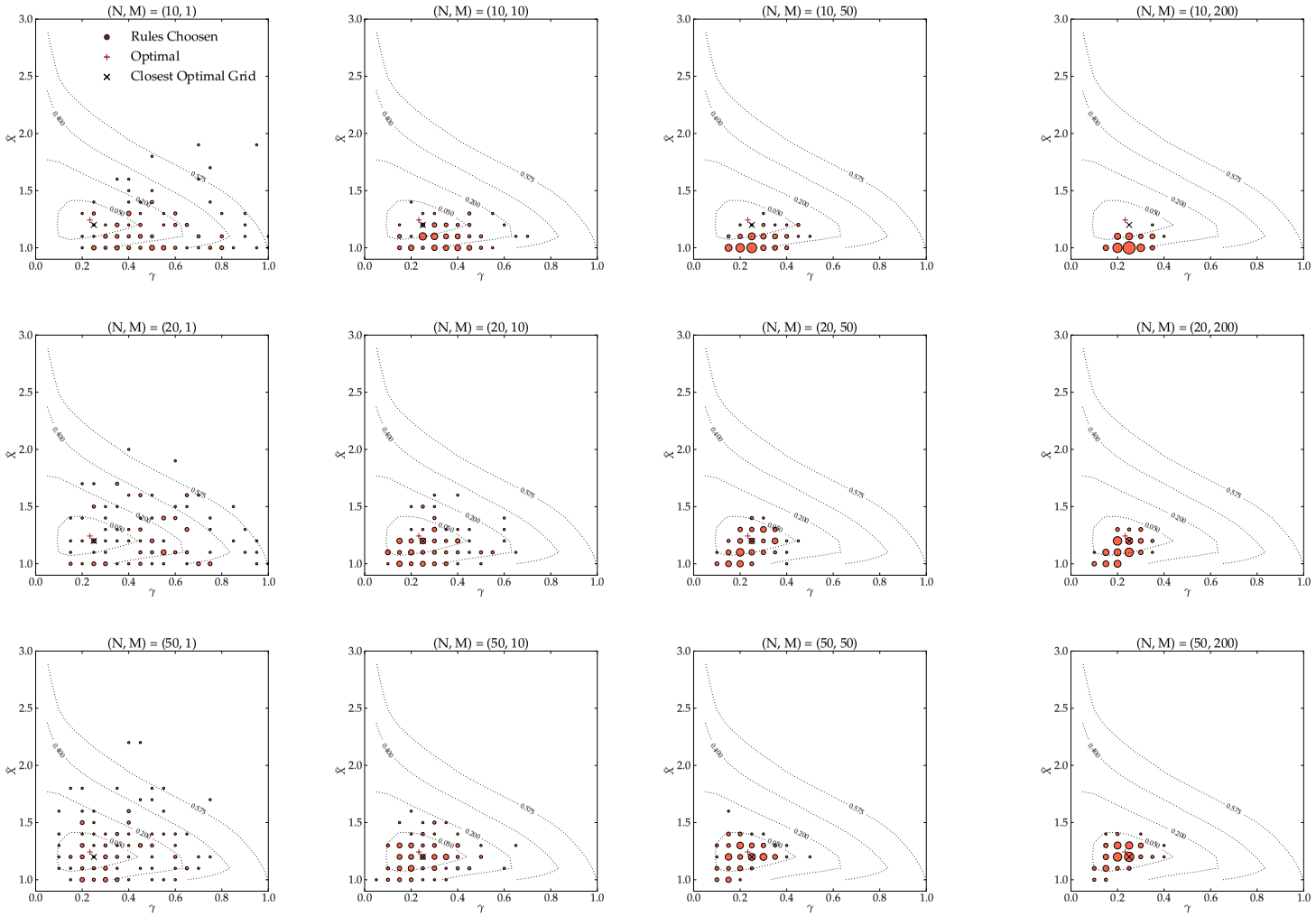


Figure D1: Distribution of Rules Selected by Agents, Original Model. Dot size Proportional to N_{agents} Choosing $(\gamma, \bar{X})$.

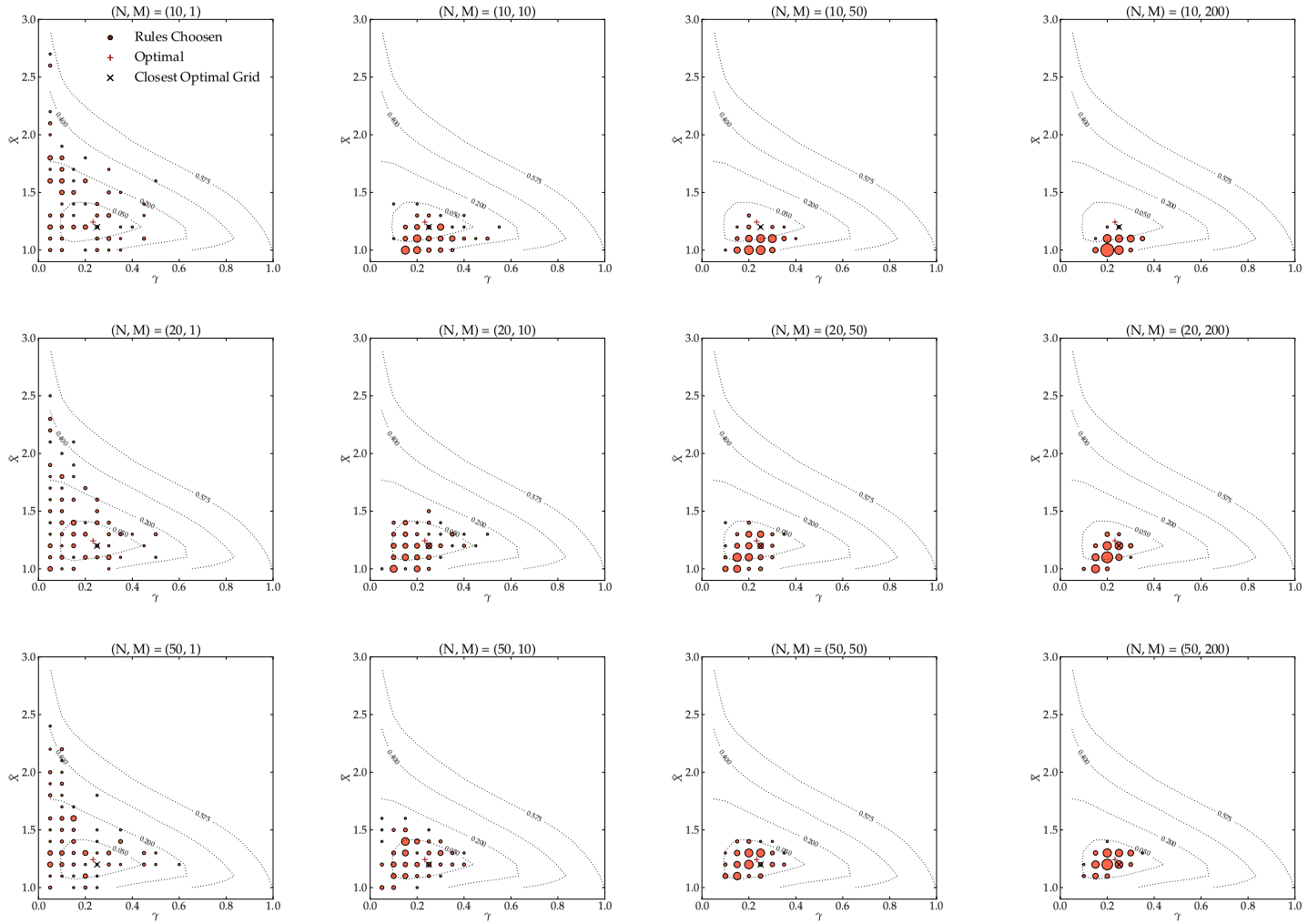


Figure D2: Distribution of Rules Chosen, Relative-Value Model. Dot Width Proportional to N_{social} choosing $(\gamma, \bar{X})$.

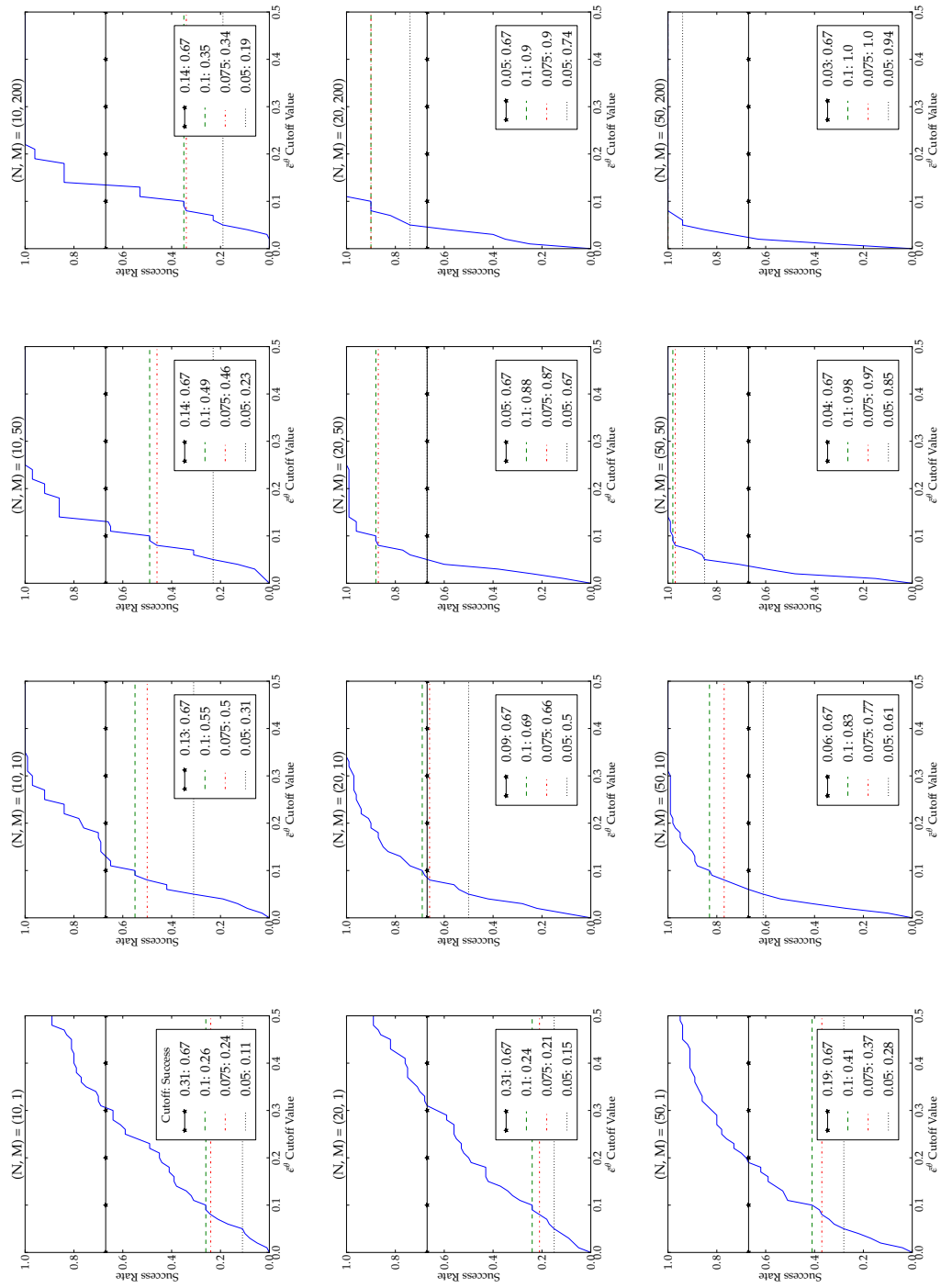


Figure D3: Fraction Successful, Original Model

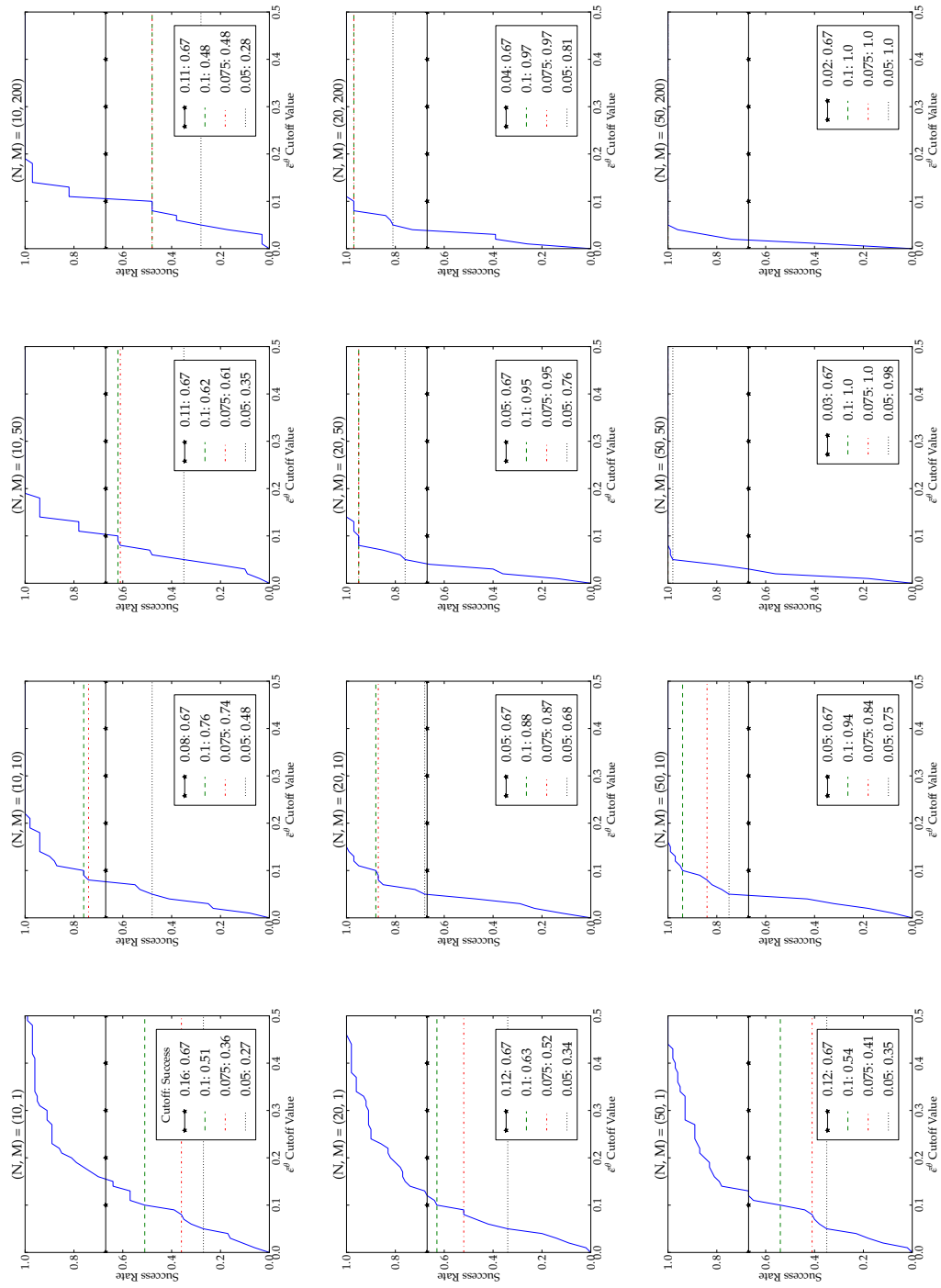


Figure D4: Fraction Successful: Relative-Value Model

Bibliography

Adjemian, S., Bastani, H., Juillard, M., Mihoubi, F., Perendia, G., Ratto, M., and Villemot, S. (2011). Dynare: Reference manual, version 4. Technical report, Dynare Working Papers 1, CEPREMAP.

Allen, T. W. and Carroll, C. D. (2001). Individual learning about consumption. *Macroeconomic Dynamics*, 5(02):255–271.

Anufriev, M. and Branch, W. A. (2009). Introduction to special issue on complexity in economics and finance. *Journal of Economic Dynamics and Control*, 33(5):1019–1022.

Aruoba, S. B. and Fernández-Villaverde, J. (2014). A comparison of programming languages in economics. Technical report, National Bureau of Economic Research.

Assenza, T., Gatti, D. D., and Semmler, W. (2013). Introduction to the special issue on rethinking policies when heterogeneity matters. *Journal of Economic Dynamics and Control*, 8(37):1401–1402.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Ballinger, T. P., Palumbo, M. G., and Wilcox, N. T. (2003). Precautionary saving and social learning across generations: an experiment*. *The Economic Journal*, 113(490):920–947.

Başçı, E. and Orhan, M. (2000). Reinforcement learning and dynamic optimization. *Journal of Economic and Social Research*, 2(1):39–57.

Bernanke, B. (2004). The great moderation. In *The Taylor Rule and the Transformation of Monetary Policy*. Institutions Press Publication Hoover.

Bernanke, B. S., Gertler, M., and Gilchrist, S. (1999). The financial accelerator in a quantitative business cycle framework. *Handbook of macroeconomics*, 1:1341–1393.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control, Vol. II, 4th Edition: Approximate Dynamic Programming*. Athena Scientific.

Bertsekas, D. P. (2013). Abstract dynamic programming. *Athena Scientific, Belmont, MA*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). Neuro-dynamic programming (optimization and neural computation series, 3). *Athena Scientific*, 7:15–23.

Brown, A. L., Chua, Z. E., and Camerer, C. (2009). Learning and visceral temptation in dynamic savings experiments. *Quarterly Journal of Economics*, 124(1):197–231.

Cagetti, M. (2003). Wealth accumulation over the life cycle and precautionary savings. *Journal of Business & Economic Statistics*, 21(3):339–353.

Carbone, E. and Duffy, J. (2014). Lifecycle consumption plans, social learning and external habits: Experimental evidence. *Journal of Economic Behavior & Organization*, 106:413–427.

Carroll, C. D. (1997). Buffer-stock saving and the life cycle/permanent income hypothesis*. *The Quarterly journal of economics*, 112(1):1–55.

Carroll, C. D. (2001a). Death to the log-linearized consumption euler equation!(and very poor health to the second-order approximation). *Advances in Macroeconomics*, 1(1).

Carroll, C. D. (2001b). A theory of the consumption function, with and without liquidity constraints (expanded version). Technical report, National Bureau of Economic Research.

Carroll, C. D. (2005). The epidemiology of macroeconomic expectations. *The Economy As an Evolving Complex System, III: Current Perspectives and Future Directions*, page 5.

Carroll, C. D. (2006). The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economics letters*, 91(3):312–320.

Carroll, C. D. (2012a). Implications of wealth heterogeneity for macroeconomics. Technical report, Working Papers, The Johns Hopkins University, Department of Economics.

Carroll, C. D. (2012b). Solving microeconomic dynamic stochastic optimization problems. Technical report, Lecture Notes, The Johns Hopkins University, Department of Economics.

Carroll, C. D. (2012c). Theoretical foundations of buffer stock saving. Technical report, Mimeo, The Johns Hopkins University, Department of Economics.

Carroll, C. D. (2014a). Heterogeneous agent macroeconomics: an example and an agenda. Technical report, Presentation at IMF Workshop on Computational Macroeconomics.

Carroll, C. D. (2014b). *Representing consumption and saving without a representative consumer*, volume 72 of *Measuring Economic Sustainability and Progress Studies in Income and Wealth*. National Bureau of Economic Research.

Carroll, C. D., Hall, R. E., and Zeldes, S. P. (1992). The buffer-stock theory of saving: Some macroeconomic evidence. *Brookings papers on economic activity*, pages 61–156.

Carroll, C. D., Otsuka, M., and Slacalek, J. (2011a). How large are housing and financial wealth effects? a new approach. *Journal of Money, Credit and Banking*, 43(1):55–79.

Carroll, C. D. and Samwick, A. A. (1997). The nature of precautionary wealth. *Journal of monetary Economics*, 40(1):41–71.

Carroll, C. D., Slacalek, J., and Sommer, M. (2011b). International evidence on sticky consumption growth. *Review of Economics and Statistics*, 93(4):1135–1145.

Carroll, C. D., Slacalek, J., Tokuoka, K., and White, M. N. (2015). The distribution of wealth and the marginal propensity to consume.

Chacon, S. and Straub, B. (2014). *Pro git*. Apress.

Chamley, C. (2004). *Rational herds: Economic models of social learning*. Cambridge University Press.

Chua, Z. and Camerer, C. F. (2011). Experiments on intertemporal consumption with habit formation and social learning.

Colander, D., Howitt, P., Kirman, A., Leijonhufvud, A., and Mehrling, P. (2008). Beyond dsge models: toward an empirically based macroeconomics. *The American Economic Review*, pages 236–240.

Evans, G. W. and Honkapohja, S. (2001). *Learning and expectations in macroeconomics*. Princeton University Press.

Evans, G. W. and Honkapohja, S. (2005). An interview with thomas j. sargent. *Macroeconomic Dynamics*, 9(04):561–583.

Evans, G. W. and McGough, B. (2014). Learning to optimize.

Gabaix, X. (2014). A sparsity-based model of bounded rationality. *The Quarterly Journal of Economics*, 129(4):1661–1710.

Geanakoplos, J. (2010). The leverage cycle. In *NBER Macroeconomics Annual 2009, Volume 24*, pages 1–65. University of Chicago Press.

Geanakoplos, J., Axtell, R., Farmer, D. J., Howitt, P., Conlee, B., Goldstein, J., Hendrey, M., Palmer, N. M., and Yang, C.-Y. (2012). Getting at systemic risk via an agent-based model of the housing market. *The American Economic Review*, 102(3):53–58.

Gourinchas, P.-O. and Parker, J. A. (2002). Consumption over the life cycle. *Econometrica*, 70(1):47–89.

Hommes, C. and Iori, G. (2015). Introduction special issue crises and complexity. *Journal of Economic Dynamics and Control*, 50:1 – 4. Crises and Complexity Complexity Research Initiative for Systemic InstabilitieS (CRISIS) Workshop 2013.

Houser, D., Keane, M., and McCabe, K. (2004). Behavior in a dynamic decision problem: An analysis of experimental evidence using a bayesian type classification algorithm. *Econometrica*, 72(3):781–822.

Howitt, P. (2013). Getting at systemic risk via an agent-based model of the housing market. Technical report, Presentation at Macro Financial Modeling Meeting May 2-3.

Howitt, P. and Özak, Ö. (2014). Adaptive consumption behavior. *Journal of Economic Dynamics and Control*, 39:37–61.

Hull, I. (2012). Interest rate rules and mortgage default.

Hyndman, R. J. and Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600.

Jirnyi, A. and Lepetyuk, V. (2011). A reinforcement learning approach to solving incomplete market models with aggregate uncertainty. *Available at SSRN 1832745*.

Kahneman, D. and Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, pages 263–291.

Krusell, P. and Smith, A. A. (1996). Rules of thumb in macroeconomic equilibrium a quantitative analysis. *Journal of Economic Dynamics and Control*, 20(4):527–558.

Krusell, P. and Smith Jr, A. (1998). Income and wealth heterogeneity in the macroeconomy. *Journal of Political Economy*, 106(5):867–896.

LeBaron, B. (2012). Heterogeneous gain learning and the dynamics of asset prices. *Journal of Economic Behavior & Organization*, 83(3):424–445.

Lettau, M. and Uhlig, H. (1999). Rules of thumb versus dynamic programming. *American Economic Review*, pages 148–174.

Ljungqvist, L. and Sargent, T. (2012). Recursive macroeconomic theory.

Loomes, G. and Sugden, R. (1982). Regret theory: An alternative theory of rational choice under uncertainty. *The economic journal*, pages 805–824.

Lucas, R. E. (1976). Econometric policy evaluation: A critique. In *Carnegie-Rochester conference series on public policy*, volume 1, pages 19–46. Elsevier.

Lucas Jr, R. E. and Stokey, N. L. (1989). *Recursive methods in economic dynamics*. Harvard University Press.

Mankiw, N. G. (2006). The macroeconomist as scientist and engineer. *Journal of Economic Perspectives*, 20(4):29–46.

Mannor, S. and Tsitsiklis, J. N. (2004). The sample complexity of exploration in the multi-armed bandit problem. *The Journal of Machine Learning Research*, 5:623–648.

Mas-Colell, A., Whinston, M. D., and Gibbons, R. (1995). Microeconomic theory.

Özak, Ö. (2014). Optimal consumption under uncertainty, liquidity constraints, and bounded rationality. *Journal of Economic Dynamics and Control*, 39:237–254.

Pál, J. and Stachurski, J. (2013). Fitted value function iteration with probability one contractions. *Journal of Economic Dynamics and Control*, 37(1):251–264.

Peterman, W., Lakdawala, A., and Cwik, T. (2015). The distributional effects of monetary policy in a life cycle model. Technical report, Presentation, Computing in Economics and Finance Conference, Society of Computational Economics, Taipei, Taiwan.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons.

Ram, Y. and Hadany, L. (2015). The probability of improvement in fishers geometric model: A probabilistic approach. *Theoretical population biology*, 99:1–6.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Sargent, T. and Stachurski, J. (2015). Quantitative economics with python. Technical report, Lecture Notes.

Sargent, T. J. (1993). Bounded rationality in macroeconomics: The arne ryde memorial lectures. *OUP Catalogue*.

Schelling, T. C. (2006). *Micromotives and macrobehavior*. WW Norton & Company.

Shalizi, C. R. (2015). Advanced data analysis from an elementary point of view. Technical report.

Sheffrin, S. M. (1996). *Rational expectations*. Cambridge University Press.

Sheppard, K. (2014). Introduction to python for econometrics, statistics and numerical analysis: Second edition. Technical report, Mimeo, University of Oxford, Department of Economics.

Simon, H. A. (1996). *The sciences of the artificial*, volume 136. MIT press.

Sims, C. A. (1980). Macroeconomics and reality. *Econometrica*, pages 1–48.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158.

Sinitskaya, E. and Tesfatsion, L. (2014). Macroeconomies as constructively rational games. Technical report, Mimeo, Iowa State University, Department of Economics.

Stachurski, J. (2009). *Economic dynamics: theory and computation*. MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press Cambridge.

Tesfatsion, L. and Judd, K. L. (2006). *Handbook of computational economics: agent-based computational economics*, volume 2. Elsevier.

Vanderbilt, T. (2013). Unhappy truckers and other algorithmic problems. *Nautilus*. [Online; posted 18-July-2013].

Woodford, M. (1999). Revolution and evolution in twentieth-century macroeconomics. In *Conference paper: Frontiers of the Mind in the Twentieth-First Century, US Library of Congress, Washington DC*.

Yıldızoğlu, M., Sénégas, M.-A., Salle, I., and Zumpe, M. (2014). Learning the optimal buffer-stock consumption rule of carroll. *Macroeconomic Dynamics*, 18(04):727–752.

Young, H. P. (2009). Innovation diffusion in heterogeneous populations: Contagion, social influence, and social learning. *The American economic review*, pages 1899–1924.

Curriculum Vitae

Nathan Palmer is a Ph.D candidate in the department of Computational Social Science at George Mason University. He was employed there as a Research Assistant building agent-based models of the economy, the financial market and the housing market and is currently employed as a Research Intern at the Office of Financial Research. He earned a M.A. in Economics from Boston University in 2011 where he specialized in macroeconomics and macro-finance. Before that he worked as a Research Assistant at the Federal Reserve Board of Governors in Washington, DC. He received his Bachelor in Computer Science from Trinity University in San Antonio, Texas in 2005.