SVERIGES RIKSBANK
WORKING PAPER SERIES
276

Approximate dynamic programming with post-decision states as a solution method for dynamic economic models

Isaiah Hull

September 2013

WORKING PAPERS ARE OBTAINABLE FROM

Sveriges Riksbank • Information Riksbank • SE-103 37 Stockholm
Fax international: +46 8 787 05 26
Telephone international: +46 8 787 01 00
E-mail: info@riksbank.se

The Working Paper series presents reports on matters in
the sphere of activities of the Riksbank that are considered
to be of interest to a wider public.
The papers are to be regarded as reports on ongoing studies
and the authors will be pleased to receive comments.

Approximate dynamic programming with post-decision states as a solution method for dynamic economic models

Isaiah Hull †

Sveriges Riksbank Working Paper Series

No. 276

September 2013

Abstract

I introduce and evaluate a new stochastic simulation method for dynamic economic models. It is based on recent work in the operations research and engineering literatures (Van Roy et. al, 1997; Powell, 2007; Bertsekas, 2011). The baseline method involves rewriting the household's dynamic program in terms of post-decision states. This makes it possible to choose controls optimally without computing an expectation. I add a subroutine to the original algorithm that updates the values of states not visited frequently on the simulation path; and adopt a stochastic stepsize that efficiently weights information. Finally, I modify the algorithm to exploit GPU computing.

Keywords: Numerical Solutions, Approximations, Heterogeneous Agents, Nonlinear Numerical Solutions, Dynamic Programming

JEL Classification: C60, C61, C63, D52

The views expressed in this paper are solely the responsibility of the author and should not be interpreted as reflecting the views of the Executive Board of Sveriges Riksbank.

†Research Division, Sveriges Riksbank, SE-103 37, Stockholm, Sweden. Email: isaiah.hull@riksbank.se.

1 Introduction

In this paper, I introduce and evaluate a new stochastic simulation method. It is based on dynamic programming, rather than the parameterization of expectational equations; and is similar to value function iteration (VFI), but does not require agents to explicitly compute an expectation over future state values when choosing controls. The baseline method, referred to as approximate dynamic programming with post decision states (ADP-POST), originates in the operations research and engineering literatures, and has been shown to substantially reduce convergence times in high dimensional problems for a given level of accuracy. I use this method to solve dynamic economic problems, focusing primarily on DSGE model applications. I start by demonstrating how the algorithm can be modified to yield performance gains over common global solution methods in representative agent models. I then show how these gains grow when the approach is modified and used to solve a model with many heterogeneous agents.

The algorithms presented in this paper are built around a post-decision state version of the Bellman equation, rather than the pre-decision state version. This approach makes it possible to solve two classes of models that are typically computationally intractable: 1) models with many continuous state variables; and 2) models with several correlated exogenous processes. However, the formal tests will focus on the former, since the latter primarily presents a programming–and not convergence time–challenge.

The baseline method used in this paper was introduced by Van Roy et. al (1997) in a neuro-dynamic programming application. There, it was used to solve a retail inventory management problem with 33 state variables. Standard dynamic programming methods, such as value function iteration (infinite horizon) and backwards recursion (finite horizon) could not feasibly solve problems with such large state spaces. Even if all of the variables were discretized into the coarsest possible grid, it would still contain 2^{33} (8.59 billion) nodes. In comparison, the neuro-dynamic programming algorithm they constructed around post-decision states was not only solvable, but delivered decision rules that yielded a 10% cost reduction over the standard, heuristic method.

This technique first entered the economics literature though Judd (1998), which explained how to construct the post-decision state Bellman equation. However, Judd (1998) did not exploit this method to explicitly construct solution algorithms. Similarly, Barto and Sutton (1998) introduced the technique to the reinforcement learning literature, but did so originally without any substantial applications. Rather, it was used primarily as a strategy for absorbing

the action space into the state space.[1]

Powell (2007) developed the method introduced by Van Roy et. al (1997) into a set of solution algorithms for high dimensionality optimization problems. However, all of the algorithms in Powell (2007) were designed for a partial equilibrium context in operations research; and most focused on inventory management applications. In problems of this variety, an agent faces random demands for a product and must choose how much inventory to hold. Using a post-decision state version of the Bellman equation works particularly well for this class of problems because the post-decision state is simply the inventory carried into the period, less the realization of demand. Outside of portfolio allocation problems, dynamic programming problems in economics do not typically take this form, and subsequently will not be able to achieve computational gains by collapsing the action space into the state space. For this reason, it is difficult to generalize what has been found in other literatures about the performance of the baseline algorithm to dynamic economic models.

Within operations research, Powell (2012) provides the most recent survey of work that has been done using post-decision state dynamic programming algorithms. Papers such as Maxwell (2011) and Simao et. al (2009) use the post-decision state formulation to solve very high dimensional problems, such as ambulance deployment and large-scale fleet management. Papers such as Powell and Ryzhov (2010) extend the initial algorithm by demonstrating how Bayesian methods can be used to update the value function.

Within economics, techniques such as the Parameterized Expectations Approach (PEA), developed by Sargent (1987), Marcet (1988), and Den Haan and Marcet (1990); and Projection Methods (PM), introduced by Judd (1992) and McGrattan (1999), provide more commonly used strategies for solving optimization problems with large state spaces. More recent work, such as Maliar and Maliar (2005) and Judd, Maliar, and Maliar (2011), demonstrates how to improve the stability and speed of such approaches.

In this paper, I use the post-decision state Bellman equation as a starting point for a larger algorithm that is designed to solve dynamic economic models. The benefit of constructing the dynamic programming problem around post-decision states is two-fold: first, it makes it possible to avoid computing expectations explicitly[2]; and second, it permits agents to make decisions

[1] In many dynamic programming applications, the endogenous component of the state space is determined by the realizations of the shock at the start of the period and the controls selected thereafter. In some of these applications, the dimensionality of the problem can be reduced simply by eliminating the choice of controls; and instead considering only the result of those choices (i.e. the end-of-period endogenous states).

[2] In fact, the expectation will be computed implicitly during the smoothing step, which adds no additional computational time.

based exclusively on information gained about the value of states visited on the simulation path, which substantially reduces the dimensionality of the choice problem. In addition to this, I make several modifications to the original algorithm in this paper to improve performance in the context of dynamic economic models.

The first modification adds a Search-Then-Converge (STC) subroutine (Darken and Moody, 1992) to update the algorithm's stepsize. However, instead of using the iteration count to update the the stepsize, as in the original application, I use the ratio of the the number of times a state has been visited on the simulation path to the iteration count. Since states in the interior of the ergodic set are visited many times within an iteration, their step sizes rapidly decline, making additional visits uninformative; however, states near the edges of the ergodic set will be able to retain larger state-specific stepsizes, allowing new visits to remain informative. Note that this modification is particularly important because the stepsize plays a dual role in algorithms based on post-decision states: it not only fosters convergence, but also implicitly computes the expectation over future states.

The second modification updates unvisited states and infrequently visited states near the edges of the ergodic set. This approach first determines a subset of the ergodic set over which the value function or decision rules are well-behaved. Information contained in this region is then used to infer properties of the decision rules over the entire state space. I do this by performing a weighted least squares regression of the decision-rule implied endogenous states on lagged state variables. I use the relative frequencies with which states were visited in the simulations as the weights. This captures the fact that infrequently visited regions of the state space are likely to have noisy state value estimates; and should be assigned lower weights.

Finally, I outline a third modification that can be used with the baseline algorithm. This modification involves placing structure on how information is shared across agents within the model; and then exploiting that structure to parallelize the solution algorithm and solve parts of it on separate GPU cores. Since the modification is not entirely specific to ADP-POST (and, thus, could be applied to the competing algorithms, such as PEA), I do not incorporate it into the accuracy tests in this paper. I do, however, discuss one application and some associated results to demonstrate its effectiveness. Furthermore, I explain how the information-sharing structure can be imposed in classes of models not tested explicitly in this paper, such as incomplete markets models with aggregate uncertainty, as in Krusell and Smith (1998).

I find that the modified version of ADP-POST introduced in this paper (hereafter, MADP-POST) is faster than both VFI and PM for a given level of accuracy. In some applications in this paper, for instance, VFI's run time is more than 600 times that of MADP-POST's.

Furthermore, as measured by static and dynamic Euler equation residuals, the modified version of ADP-POST's accuracy is generally of the same order of magnitude as VFI and PM. The only algorithm that outperforms MADP-POST in some of this paper's tests is PEA. However, unlike MADP-POST, PEA is difficult to parameterize in sophisticated applications; and is prone to instability in complex models (Judd, Maliar, and Maliar, 2011). Additionally, as is shown in a heterogeneous agents model application, the case for MADP-POST improves further when it is combined with the information sharing algorithm discussed above and GPU-computing. In particular, the modified algorithm (MADP-IS-GPU) converges faster on a grid of 200,000 nodes than either VFI, PEA, PM, or ADP-POST converges on a grid of 10,000 nodes. Furthermore, it does so while remaining highly accurate and stable.

The paper will proceed as follows. In Section 2, I will describe the ADP-POST algorithm and all relevant modifications made for this paper. In Section 3, I will provide brief overviews of value function iteration (VFI), the parameterized expectations approach (PEA), and projection methods (PM)–the three categories of solution method I will use for comparison. In Section 4, I will explain how the accuracy of the different solution methods will be tested. In Section 5, I will present benchmark models that will be solved using the aforementioned algorithms and also discuss the results. And finally, in Section 6, I will conclude.

2 The Algorithm

2.1 The Pre-Decision State Bellman Equation

I will start by describing the formulation of the Bellman Equation with pre-decision state variables; and will then move on to the approach for post-decision state variables. This construction of the Bellman Equation will focus on an infinitely-lived agent who faces a deterministic choice problem, which requires her to choose a set of controls, c_t. A set of states, s_t, summarizes all information needed to make a choice at time t. Combined with c_t, s_t pins down the future state according to a transition function: $s_{t+1} = T(c_t, s_t)$. Finally, a set of constraints, $c_t \in \Gamma(s_t)$, limits the choice of controls. After applying the Principle of Optimality as described in Bellman (1954), the value of being in the initial state can be written as follows:

$$\begin{aligned} V(s_0) &= max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u(c_t) \\ \text{s.t. } s_{t+1} &= T(c_t, s_t),\ c_t \in \Gamma(s_t),\ \forall t = 0, 1, ... \end{aligned} \tag{1}$$

That is, the value of being in the initial, pre-decision state (i.e. before controls are selected) is the maximum, discounted lifetime utility that can be achieved through an infinite sequence of

control choices. Note that computing $V(s_0)$ is a difficult task because it depends on this infinite sequence. However, Bellman (1954) showed that we could simplify the problem by rewriting $V(s_0)$ as follows:

$$\begin{aligned} V(s_0) &= max_{c_0}\{u(c_0) + [max_{\{c_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^t u(c_t)]\} \\ \text{s.t. } & s_{t+1} = T(c_t, s_t),\ c_t \in \Gamma(s_t),\ \forall t = 0, 1, ... \end{aligned} \tag{2}$$

Now, notice that we can exploit the recursive property of this equation to rewrite it again:

$$\begin{aligned} V(s_0) &= max_{c_0}\{u(c_0) + \beta \underbrace{[max_{\{c_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u(c_{t+1})]}_{V(s_1)=V(T(s_0,c_0))}\} \\ \text{s.t. } & s_{t+1} = T(c_t, s_t),\ c_t \in \Gamma(s_t),\ \forall t = 0, 1, ... \end{aligned} \tag{3}$$

$$\rightarrow V(s_0) = max_{c_0} u(c_0) + \beta V(s_1) \quad \text{s.t. } s_1 = T(c_0, s_0),\ c_0 \in \Gamma(s_0) \tag{4}$$

This equation–the Bellman equation–is much simpler in the sense that it permits us to break a multi-period decision problem down into a series of smaller, tractable steps, where only one set of controls must be selected at a time. However, it is more complicated in the sense that it now requires knowledge of an unknown function, V. Analytical dynamic programming solves this problem by recovering the algebraic expression for the value function, V, but is limited in application. In contrast, numerical dynamic programming constructs an approximation of V and is applied to a wide variety of multi-period choice problems.

Next, I will augment the household's problem to incorporate exogenous processes, which are denoted by $x_{t+1} = f(x_t, \epsilon_{t+1})$. The subscript t+1 indicates that the exogenous shocks arrive at the start of period t+1 and before time t+1 controls are chosen. Note that the joint transition function for states now captures the impact of exogenous shocks: $s_{t+1} = T(c_t, s_t, x_{t-1}, \epsilon_t)$.[3] Furthermore, I assume that the feasible set of control choices also depends on the exogenous process: $c_t \in \Gamma(s_t, x_t)$. The value of being in the initial state may now be written as follows:

$$\begin{aligned} V(s_0, x_0) &= max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t E[u(c_t)|s_0, x_0] \\ \text{s.t. } & s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} = f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{5}$$

[3]This transition function is written as a joint process; however, it is important to note that s_{t+1} depends on x_t and s_t, but x_{t+1} is independent of s_t and c_t.

Again, I will exploit the recursive nature of $V(s_0, x_0)$ to rewrite it as follows:

$$\begin{aligned} V(s_0, x_0) = max_{c_0}\{u(c_0) + \beta \underbrace{[max_{\{c_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t E[u(c_{t+1})|s_0, x_0]]}_{E[V(s_1,x_1)]=E[V(T(c_0,s_0,x_0))]}\} \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} = f(x_t, \epsilon_{t+1})\ \forall t = 0, 1, ... \end{aligned} \tag{6}$$

This yields the stochastic version of the Bellman equation:

$$\begin{aligned} V(s_0, x_0) = max_{c_0} u(c_0) + \beta E[V(s_1, x_1)] \\ \text{s.t. } s_1 = T(c_0, s_0, x_0),\ c_1 \in \Gamma(s_0, x_0),\ x_1 = f(x_0, \epsilon_1) \end{aligned} \tag{7}$$

More generally, the Bellman equation can be written as follows:

$$\begin{aligned} V(s_t, x_t) = max_{c_t} u(c_t) + \beta E[V(s_{t+1}, x_{t+1})] \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} = f(x_t, \epsilon_{t+1}) \end{aligned} \tag{8}$$

The derivations above yield both the deterministic and stochastic versions of the standard Bellman Equation in terms of pre-decision state variables. In the next section, I will explain how to rewrite the Bellman equation in terms of post-decision states. This mathematically simple modification will afford a tremendous amount of modeling flexibility; and will render otherwise intractable dynamic economic models solvable.

2.2 The Post-Decision State Bellman Equation

In this section, I will construct the Bellman equation in terms of post-decision state variables, roughly following (but also expanding on) the treatment of this subject in Powell (2007) and Bertsekas (2011). Additionally, I will focus on building an adaptation that is well-suited to the timing conventions and model properties of dynamic economic models, rather than operations research or engineering applications. This exposition will begin with a comparison of pre-decision and post-decision states to make this distinction clear. A pre-decision state consists of endogenous state variables determined at the end of the previous period, s_t, and realizations of the exogenous process at the start of this period, x_t. This state is "pre-decision" because it is determined entirely before the time t controls, c_t, are selected. In contrast, the post-decision state consists of the endogenous state variables determined at the end of period t, s_{t+1}, and exogenous processes realized at the start of period t, x_t. Recall from the construction of the Bellman equation in the previous section that $s_{t+1} = T(c_t, s_t, x_t)$. That is, near the start of time t, s_t and x_t are pinned down. Once c_t is chosen, s_{t+1} will be pinned down, too. This yields the post-decision state–that is, the state immediately after we have selected controls. Figure 1 shows this relationship through the arrival of new information in discrete time model.

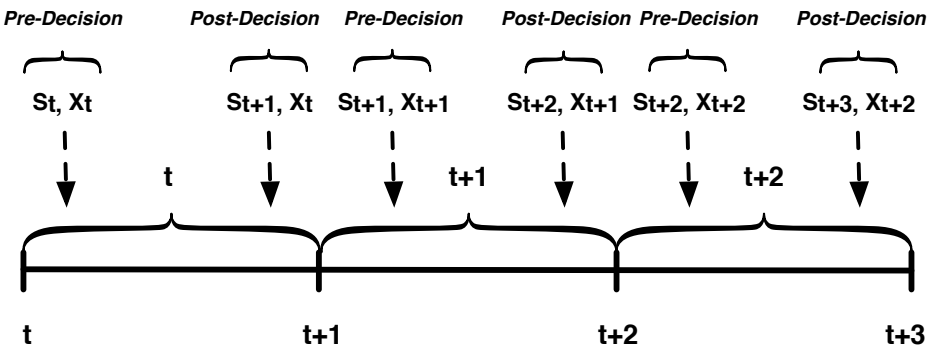

Figure 1: Timing conventions for the pre-decision and post-decision state variables.

Note that the *value* of a post-decision state is the maximum, expected, discounted utility that the agent can achieve immediately *after* controls have been selected. Using this definition, the post-decision state value function, $V^x(s_t, x_{t-1})$, can be written as follows:

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E\{max_{\{c_{t+s}\}_{s=0}^{\infty}} \sum_{s=0}^{\infty} \beta^s E[u(c_{t+s})|s_t, x_{t-1}]\} \\ \text{s.t. } s_{t+s+1} &= T(c_{t+s}, s_{t+s}, x_{t+s}),\ c_{t+s} \in \Gamma(s_{t+s}, x_{t+s}),\ x_{t+s+1} = f(x_{t+s}, \epsilon_{t+s}),\ \forall s = 0, 1, ... \end{aligned} \tag{9}$$

We will now manipulate this equation to derive useful, related properties, as well as a final expression for the post-decision state Bellman equation. We begin this process by recalling that the agent will not choose c_t until x_t has been realized. For this reason, $V^x(s_t, x_{t-1})$ can also be written as the expectation, conditional on the information set (s_t, x_{t-1}), of the maximum, expected, discounted utility that the agent will receive after x_t arrives. This is equivalent to the expected value of state (s_t, x_t), conditional on the information set (s_t, x_{t-1}):

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E[V(s_t, x_t)|s_t, x_{t-1}] \\ \text{s.t. } s_{t+1} &= T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} = f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{10}$$

Stepping forward in time, we may write the value of being in post-decision state (s_{t+1}, x_t) as follows:

$$\begin{aligned} V^x(s_{t+1}, x_t) &= E[V(s_{t+1}, x_{t+1})|s_{t+1}, x_t] \\ \text{s.t. } s_{t+2} &= T(c_{t+1}, s_{t+1}, x_{t+1}),\ c_{t+1} \in \Gamma(s_{t+1}, x_{t+1}),\ x_{t+2} = f(x_{t+1}, \epsilon_{t+2}),\ \forall t = 0, 1, ... \end{aligned} \tag{11}$$

Finally, one additional relationship is needed. It can be shown that the following holds between the pre-decision and post-decision state Bellman equations:

$$\begin{aligned} V(s_t, x_t) &= max_{c_t} u(c_t) + \beta V^x(s_{t+1}, x_t) \\ \text{s.t. } s_{t+1} &= T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} = f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{12}$$

Short proofs are given in the appendix for the claims made by equations (10) and (12).[4] These two equations, as well as equation (11), can be used to demonstrate useful relationships between the pre-decision and post-decision state Bellman equations. For instance, if (10) is substituted into (12), it yields the standard, pre-decision Bellman equation. On the other hand, if (12) is substituted into (10), it yields the following post-decision state Bellman equation:

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E\{max_{c_t} u(c_t) + \beta V^x(s_{t+1}, x_t) | s_t, x_{t-1}\} \\ \text{s.t. } s_{t+1} &= T(c_t, s_t, x_t),\ c_t = \Gamma(s_t, x_t),\ x_{t+1} \in f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{13}$$

There are two important things to note. First, the expectations operator is outside of the maximum operator. This differs from the standard, pre-decision state Bellman equation. And second, the problem within the expectation is *deterministic*.

Van Roy et. al (1997), Judd (1998), and Barto and Sutton (1998) all separately identified this alternative form of the Bellman equation. Powell (2007) and Bertsekas (2011) provided the first general descriptions of how it could be used to construct solution algorithms for operations research and engineering applications respectively. In particular, they demonstrated that it was most valuable in high-dimensional dynamic programming problems, as well as problems where the stochastic processes were complicated or unknown. Modern, computational textbook approaches to solving dynamic, stochastic general equilibrium models, such as those in Heer and Maussner (2009) and DeJong and Dave (2007), do not discuss either the post-decision state Bellman equation or the associated algorithms.

2.3 The Post-Decision State Dynamic Programming Solution Algorithm

In the previous section, I demonstrated how to construct the post-decision state Bellman equation. Next, I will discuss two algorithms that take advantage of this alternative approach to dynamic programming. Both of these solution methods fall into the family of approximate dynamic programming algorithms that exploit post-decision states. The first approach is the ADP-POST algorithm,[5] modified for use in a dynamic economic model. The second approach is a refinement that incorporates an information-sharing subroutine that exploits GPU-computing. The algorithms described in this section will follow Powell (2007) and Bertsekas (2011).

Before we construct the algorithm, it is useful to recall the form of the post-decision state

[4] As far as this author is aware, this paper provides the first proofs for these three important properties, which are referenced in Powell (2007).

[5] This is sometimes referred to as neuro-dynamic programming, reinforcement learning with post-decision states, or forward dynamic programming with post-decision states.

Bellman equation from the previous section:

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E\{max_{c_t} u(c_t) + \beta V^x(s_{t+1}, x_t)|s_t, x_{t-1}\} \\ \text{s.t. } s_{t+1} &= T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} \in f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{14}$$

Notice again that the maximization step is inside of the expectation. This property of the post-decision state Bellman equation initially appears to make things more complicated, but actually makes it substantially easier to solve certain classes of models. In particular, it will make it possible to construct algorithms that first perform the maximization step; and then compute the expectation afterwards. This first step is then combined with draws from the exogenous processes, which transition the agent to the next step; and then a smoothing step, which implicitly computes the expectation outside of the maximization step while fostering convergence. The baseline algorithm is outlined below and follows Powell (2007).

2.4 *Algorithm 1.1: ADP-POST (Infinite Horizon, Representative Agent)*

The ADP-POST algorithm will provide a foundation for all algorithms constructed in this paper.

The baseline version of the algorithm proceeds as follows:

- *Step 0:* Perform all initializations.
  - i. Initialize the "look-up table" approximation of the value function in all states. That is, for each state, $(s \in \mathbf{s}, x \in \mathbf{x})$, assign a starting value to $\overline{V}^0(s, x)$, where the superscript 0 indicates that this is an approximation taken at the 0th iteration.
  - ii. Initialize the state, $(s_0 \in \mathbf{s}, x_0 \in \mathbf{x})$. For representative agent, infinite horizon models, it may be natural to assume the the agent starts in the steady state.
  - iii. Increment the iteration counter to n=n+1.
- *Step 1:* Choose a simulation period length, T. Generate a sample of the exogenous processes, x_t, for the simulation period: 1,...,T.
- *Step 2*: For all periods, t=1,...T, perform the following three steps:
  - i. Choose controls to maximize the expression *inside* of the expectation of $V(s_t, x_{t-1})$ for a particular realization of x_t, given that $c_t \in (s_t, x_t)$:
    $\tilde{V}^n(s_t, x_{t-1}) = max_{c_t} u(c_t) + \beta \overline{V}^{n-1}(s_{t+1}, x_t)$,
  - ii. Compute the expectation by updating the value function approximation, V^{n-1}:
    $\overline{V^n}(s_t, x_{t-1}) = (1 - \alpha_{n-1})\bar{V}^{n-1}(s_t, x_{t-1}) + \alpha_{n-1}\tilde{V}^n(s_t, x_{t-1})$
    Note that α denotes the step size. We index it with n to indicate that it may change with the iteration. It may also be stochastic.

  - iii. Use the result from the maximization step, s_t, and the realization of the exogenous processes, x_{t+1}, to compute the next period's pre-decision state.

- *Step 3:* Check convergence criterion. If satisfied, go to Step 4. If not satisfied, increment n and go to Step 1.

- *Step 4:* Return the value function approximation, $V(s,x)^N$, in the form of a look-up table.

Now that the basic algorithm has been described, it will be useful to examine Steps 1 and 2 in greater detail. In Step 1, a sample of the exogenous processes is drawn. In a standard DSGE model with a representative agent, these processes might capture things like productivity shocks. Usually, it is assumed that these exogenous variables follow a continuous process, which can be approximated through discretization. Often, Tauchen (1986) is employed to construct a Markov chain and transition matrix that approximate the continuous analog. Alternatively, quadrature methods can be used to approximate the expectation.

One of the benefits of using Algorithm 1.1 is that it is possible to avoid explicitly computing expectations using either of these approaches. That is, the household's dynamic programming problem can be solved without ever employing quadrature methods or transforming the assumed continuous, exogenous processes into discrete Markov processes. To understand why this is important, consider the two examples given below.

Example 1. *Correlated Exogenous Processes.* In a model with several sectors or several assets, it is common to have two exogenous processes that comove empirically. Solving computational models will often require the assumption that processes are orthogonal for the sake of simplicity. Assuming this makes it possible to take expectations without explicitly accounting for the dependence between the processes. That is, if z and q are the exogenous variables in the household's problem, then z_{t-1} will yield the pre-shock distribution of z_t, and q_{t-1} will yield the pre-shock distribution q_t. However, if we assume that these processes are dependent, then z_t depends not only on z_{t-1}, but also on q_t. This not only requires us to use more technically sophisticated quadrature methods or discretization techniques (i.e. conditional Markov processes), but also requires us to use a higher dimensional approximation in order to capture the dependence relationship accurately.

In contrast, if Algorithm 1.1 is employed, then it is not necessary to devise a strategy for computing the expectation. Being able to simulate the exogenous process will be sufficient. This could be done, for instance, by using a copula simulator to generate series for the exogenous processes before the algorithm is even initiated. Step 2(ii) will then compute an approximation

of the expectation that improves as the simulation length increases. This suggests that constructing and solving a model with many exogenous processes that depend on each other will be no more difficult than constructing and solving a model with the same number of independent processes.

Example 2. *Exogenous Processes with Unknown Functional Form.* Consider a model in which a household may hold many classes of assets. Assume further that the returns to those assets depend on exogenous processes; and that those processes exhibit dependence. Working with a pre-decision state value function, it will be necessary to use technically sophisticated methods (i.e. copula estimation or multiple conditional Markov processes); and then make functional form assumptions about the joint distribution from which shocks are drawn. In contrast, using post-decision state variables, it is possible to run simulations with the actual asset data without ever performing the intermediate steps.

In short, examples 1 and 2 demonstrate the benefits of using the post-decision state value function: it enables us to bypass difficult expectations without ever explicitly computing them. Furthermore, in some cases, it makes it possible to avoid making any assumptions at all about the exogenous processes in the model, which can both limit model error and eliminate the need for highly sophisticated techniques.

I will now return to Step 2 in greater detail, which is the other major departure from pre-decision state dynamic programming algorithms. In particular, in substep (i), controls were selected to maximize a *deterministic* function, which is a substantial departure from what is normally done in stochastic dynamic programming problems. This has the primary benefit of reducing the dimensionality of the problem. Substep (ii) may also look unfamiliar, even though it is similar to smoothing step often used in pre-decision dynamic programming algorithms. Here, it not only plays the role of smoothing, but also implicitly computes the expected value of being in state (s_t, x_{t-1}). To understand why this is the case, notice that the maximum value of $\tilde{V}(s_t, x_{t-1})$ will depend on the realization of x_t. Thus, arriving at (s_t, x_{t-1}) for a second, third, and fourth time, but with a different realization of x_t will provide more information about the value of being in state (s_t, x_{t-1}). Furthermore, the frequency with which agents encounter certain realizations of x_t will provide information about the probabilities of realizations.

2.5 *Algorithm 1.2: ADP-POST with Information-Sharing (Infinite Horizon, Heterogeneous Agents)*

One property of ADP-POST is that it simultaneously solves the choice problem and simulates agent behavior. This differs from VFI and PM, which must first solve the problem and then simulate agent behavior. The benefit of this property is that it allows us to solve the model for many agents on different stochastic simulation paths simultaneously. Information can then be shared across agents at some specified point in the algorithm. In the case of incomplete markets models with aggregate uncertainty, the aggregation step–where the laws of motion are estimated–provides a natural, unparallelizable point at which agents may share information about the values of states visited. In the algorithm below, I demonstrate how this can be done for ADP-POST. Note that S and X denote the sets of endogenous and exogenous aggregate state variables, respectively. Furthermore, note that the algorithm is provided for the most general case, where the model contains endogenous and exogenous state variables; however, for simpler applications, S and X can be removed and the aggregation step can be eliminated.

- *Step 0-Step2(ii)*: See Algorithm 1.1.

- *Step 2(iii)*: Compute the agent-specific expectation by updating the value function approximations, V_i^{n-1}, separately for each household: $\overline{V_i^n}(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) = (1-\alpha_{n-1})\bar{V}_i^{n-1}(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) + \alpha_{n-1}\tilde{V}_i^n(S_{t-1}, X_{t-1}, s_{it}, x_{it-1})$
  Note that α denotes the step size. It is indexed with n to indicate that it may change with the iteration. Households in are indexed by i.
  iv. Average the value function approximations across households to share all available information:
  $\overline{V^n}(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) = \sum_{i \in I} \bar{V}_i^n(S_{t-1,}, X_{t-1}, s_{it}, x_{it-1})\mu_i$
  Note that I is the set of all households and μ_i is the mass of household i. For example, if there are K agents of equal mass in the model, then $\mu_i = \frac{1}{K}$, for all i. Alternatively, the distribution might be written as function of state variables.

- *Step 2(iii) - Step 4*: See Algorithm 1.1.

It is important now to gain a full understanding of the merits of Algorithm 1.2. A good place to start is by comparing it to an extension of Algorithm 1.1 with heterogeneous agents, but without information-sharing. Here, households solve their decision problems simultaneously, but do so without the benefit of what other agents have learned. If information-sharing is

imposed in a model with 10,000 heterogeneous agents, then each agent gains information about the value of 10,000 state-path trajectories after each iteration, rather than one.

Another possible approach to this problem would involve sharing information across agents within iterations. In the limit, this approach might entail using the same value function approximation for all agents. This approach has two benefits: first, it provides agents with more information about the values of states within iteration, leading to faster convergence if all other things are equal; and second, in the limiting case (one value function approximation only), it substantially reduces the amount of data that must be held in memory (down from 10,000 arrays to 1). However, there is another drawback to using this approach: it does not lend itself to parallelization. That is, it is not possible to solve each household's problem in isolation on a separate CPU or GPU core if household i's value function approximation depends on household j's approximation. For this reason, the algorithm above defers information-sharing and performs it as an end-of-iteration task. This will not prevent further parallelization, since heterogeneous agent models require aggregation and law of motion estimation at the end of each iteration, regardless of how the household problem is solved.

I have now outlined the two primary algorithms that are the focus of this paper. The first algorithm will be used to test ADP-POST relative to common global solution methods. The second algorithm can be used in more complex applications, but will only be tested once and on a large grid to demonstrate its effectiveness. I will then discuss two modifications I make to the original ADP-POST algorithm to improve its accuracy and stability. I will also discuss an extension of the ADP-POST algorithm in the appendix, which will be useful for finite horizon models, including OLG macro models with many heterogeneous agents. After those subsections, I review global solution methods that are common in the DSGE literature, which will be used as performance benchmarks. In particular, I will focus on common methods for infinite horizon models, including value function iteration, the parameterized expectations approach, and projection methods.

2.5.1 *Modified Search-Then-Converge (STC) Stepsize*

Stepsize selection is an important component of pre-decision state dynamic programming algorithms; however, it is an even more important choice for the ADP-POST family of algorithms. This is because the stepsize not only fosters convergence, but also implicitly computes the expectation over the values of future states. Poor stepsize selection can result in either ignoring important information about states or assigning too much weight to the value of the most recently visited state.

In general, a well-constructed stepsize for ADP-POST should remain large during early iterations, but decline sharply as more of the state space is visited on the simulation path, making new observations less informative. A commonly used stepsize with such properties is Darken and Moody's (1992) Search-Then-Converge (STC) routine, which is updated as follows:

$$\alpha_n = \alpha_0 \left(\frac{1 + \frac{\beta}{\alpha_0} \frac{n}{\tau}}{1 + \frac{\beta}{\alpha_0} \frac{n}{\tau} + \frac{n^2}{\tau}} \right) \tag{15}$$

In the above equation, n is the outer iteration counter, α_n is the stepsize at iteration n, and β and τ are parameters that can be tuned to adjust the onset and duration of the "search" and "converge" phases. It is important to note that the above equation implies that a uniform stepsize should be applied to all value function updates within a given iteration. This suggests that information about a state that has never been visited previously should be treated as if it were exactly as informative as information about a state that has been visited thousands of times on the simulation path. This is particularly problematic, since infrequently visited states will tend to be reached in later iterations, where small stepsizes will be applied. This will tend to bias the values of infrequently visited states towards their initializations.

I modify the STC algorithm by performing value function updates during the stochastic simulation and by using state-specific stepsizes. I start by constructing an associated matrix, Q, of state-visit counts. The element, $q_{i,j} \in Q$, is the number of times state (i,j) has been visited in all previous simulation steps, including simulations in earlier iterations. Next, I replace n in equation (15) with $q_{i,j}/n$. Note that this functional form normalizes the state-visit count by the total number of outer iterations. This will create a stepsize that assimilates information depending on the relative frequency with which states have been visited. States within a narrow area around the steady state will be visited frequently and will quickly become uninformative; however, state visits further away will still remain informative in later iterations.

It is important to note that increasing the size of the state space will impact how this modified version of the STC stepsize algorithm works. As an example, consider the case where the simulation length is T, the number of capital states is $\bar{k}$, and the number of productivity states is $\bar{z}$. On average, $q_{i,j}$ will grow at rate $\frac{T}{k\bar{z}}$. Thus, if $\bar{z}$ or $\bar{k}$ is increased, but T is left unchanged, then the onset of the the convergence phase will be delayed. Furthermore, if $\bar{z}\bar{k} > T$, then the stepsize will tend to grow over time, preventing convergence; and, finally, if $\bar{z}\bar{k}$ is larger than, but close to T, then nodes far away from the steady state will have a low $q_{i,j}$, and may have large stepsizes applied to them, even if frequently visited states do not.

Figure 2 provides a visual comparison of the modified and unmodified STC stepsize algorithms for the first 15,000 periods of a simulation. The primary difference between the two

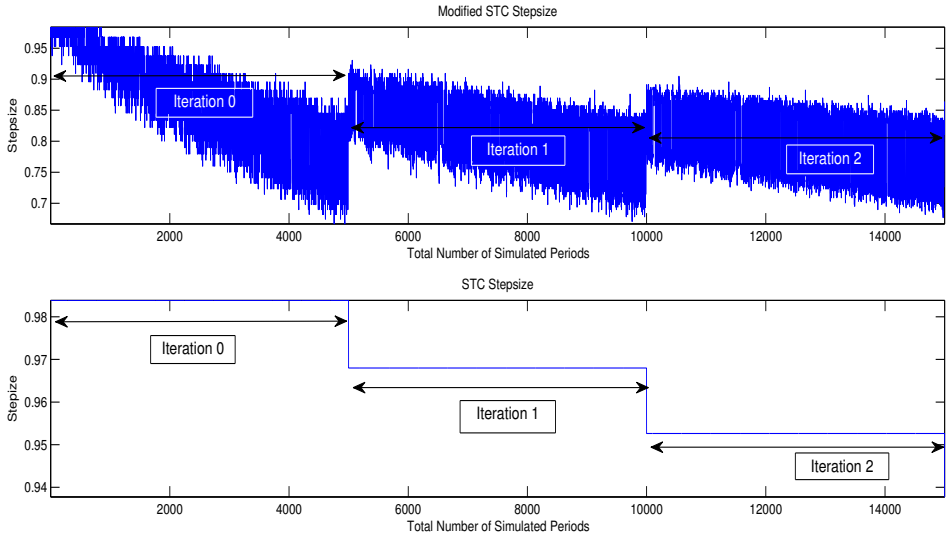

Figure 2: Figure 2: STC and Modified STC

algorithms is that there is no variation in stepsize within the outer iteration of the unmodified STC algorithm. To the contrary, the stepsize in modified STC is updated in every simulation period and depends on $q_{i,j}$. Notice that this allows for a substantial amount of flexibility within an outer iteration: we can simultaneously apply small stepsizes to frequently visited states and large stepsizes to infrequently visited states, which permits us to lower the average stepsize quickly without ignoring important information. Finally, note that Figure 2 shows stepsizes for the case where $\bar{z}\bar{k}$ is close to T, which illustrates the point from the previous paragraph: after iteration 0, the modified STC stepsize decreases slowly across outer iterations.

2.5.2 *Exploiting Well-Behaved Regions*

This component of the MADP-POST algorithm consists of two steps. In the first step, a well-behaved region of either the decision rules or the value function is identified. In the second step, that well-behaved region is used to draw inferences about the shape of the decision rules or value function outside of that region.

Step 1: Identify the Well-Behaved Region

The first step begins by identifying all relevant discontinuities in the decision rules. These discontinuities will generally fall into one of two categories. The first category involves a binary choice (e.g. default) or a constraint. This category of discontinuities generates no substantial challenges for this step of the ADP-POST algorithm because they occur at known locations in

the value function. The other group may arise from subtle modeling features that introduce a discontinuity at an unknown location in the value function. This category of discontinuity can introduce problems if it appears near the edges of the ergodic set and is infrequently visited in simulations. For this reason, it is important to identify the plausible locations of such discontinuities if they are present in a model; and to adjust the algorithm accordingly, so that the region around the discontinuity is reached in simulations.

Next, if all discontinuities are determined to be of the first type–which will be true for many classes of problems–then we may proceed by solving the model with the unmodified ADP-POST algorithm. We can then extract the value function and implied decision rules; and use them to define a well-behaved region. The following can be used to identify the boundaries of the well-behaved region:

- The Q matrix.
- The monotonicity of the value function or decision rules in continuous arguments, such as capital.
- The monotonicity of the value function or decision rules in discrete arguments, such as income endowment shocks.
- The concavity of the value function in capital–and, more generally, real asset holdings.
- The slope of conditional decision rules for a state variable.

The Q matrix provides us with the frequency with which each state was visited on the simulation path. Initially, using Q may seem to be an efficient way to determine the "well-defined" region; however, this approach suffers from two problems. First, the decline in visit frequency is likely to occur continuously over some dimension of a decision rule. This will make it difficult to identify an exact threshold between areas that have been visited "too infrequently" and areas that have been visited "enough." For instance, using 0 state-visits as a threshold will not yield accurate results. This is because regions with a small, but positive number of state-visits will not only be highly inaccurate, but will be biased in the direction of the initialization. Furthermore, this approach is problematic because different state-visits receive different weights, depending on when they occurred. Thus, the Q matrix may provide some useful information about identifying the boundaries, but it should not be used alone.

We will now consider the monotonicity of the value function or decision rules in continuous arguments, such as capital. Often, if there are no discontinuities of the second type in a model, we will know a priori that the value function is weakly monotonic in certain continuous

arguments. In the case of capital, for instance, we know that entering a period with a higher capital stock will always permit the agent to achieve a higher path of consumption streams. This suggests that the value function should be monotonically increasing in capital, conditional on all other states. This provides us with a simple test: if a violation of monotonicity occurs in a region with no discontinuity, then this part of the value function is outside of the well-behaved region. A similar test can be applied to the decision rules. It should be noted, however, that such a test may be too weak, since it will not detect a boundary, but will instead identify that a node is outside of that boundary.

A similar test can be performed for discrete arguments, such as unemployment shocks or discretized income shocks. If, for instance, a model has 5 discrete income states, we might expect the value function to be monotonic in those states. This is because a higher income state will allow the agent to achieve a higher path of consumption streams, conditional on a particular set of state variables. This suggests that we can check if the monotonicity relation holds at each point in the state space; and rule out parts where it does not.

Another test we can perform relies on the concavity of the value function in certain arguments, such as capital. In the absence of the second type of discontinuity, this concavity will often arise from the shape of the utility function and discounting. To understand how this works, consider an evenly-spaced capital grid with kn nodes: $k \in [k_1, ..., k_{kn}]$. If the value function is concave in capital, then the value of k_2 should be closer to k_3 than it is to k_1. This is a stronger requirement than monotonicity and will be useful if we wish to conservatively define the well-behaved region.

Finally, the slope of a conditional decision rule can provide useful information about whether or not an area falls within the well-behaved region. In particular, if we compute the slope of k', plotted against k, for a given set of other states, we may find that it is implausible in certain places, given the properties of the model. For instance, if the decision rule implies that an increase in period t capital of 1 unit leads to an increase in period t+1 capital of 2 units, we may rule this out as implausible. More generally, we may impose the following condition for a given value of γ:

$$(k'(k_{j+1}) - k'(k_j)) < \gamma(k_{j+1} - k_j) \tag{16}$$

If this condition is violated, then we may state that the associated node is not within the well-behaved region. We can then exclude that node from Step 2, as well as any nodes further away from the steady state along that dimension.

Step 2: Update the Value Function or Decision Rules

We have now defined a "well-behaved" region of either the decision rules or value function. Next, we will exploit the properties of this region to update the values of states outside of it. One simple way in which we can do this is to regress the values of states in the well-behaved region on polynomial transformations of the the state variables. For simple problems, an exponential polynomial of the natural log of states may be sufficient. For more complicated problems, it may be necessary to use high order, orthogonal polynomial transformations to capture the relevant nonlinearities. Note that these same techniques can alternatively be used to estimate the decision rules, rather than the values of states.

The baseline method described above can be modified in two ways to yield better results. The first modification entails using WLS, rather than OLS. Recall that the WLS estimator is given as follows:

$$\beta_{WLS} = (X'\Omega^{-1}X)^{-1}X\Omega^{-1}Y \tag{17}$$

In the above equation, $\Omega^{-1} = P'P$, where P is a diagonal matrix of weights. Since the precision of state value estimates will depend on the frequency with which those states were visited on the simulation path, we may use a transformation of the Q matrix to construct weights for WLS. In particular, we will let $w_{i,j}$, the weight associated with state (i,j), be equal to the proportion of total state visits in the (i,j) element of Q:

$$w_{i,j} = \frac{q_{i,j}}{\sum_{r\in I}\sum_{s\in J} q_{r,s}} \tag{18}$$

The second modification involves iterating over the estimation stage, rather than performing it after unmodified ADP-POST converges. Here, it is recommended that you use a numerically stable method of updating the parameters, such as a Tikhonov regularization:

$$\beta_{RLS-TV} = (X'X + \eta I_n)^{-1}X'Y \tag{19}$$

Note that η is a small scalar. For an overview of the Tikhonov regularization, as well as other numerically stable methods for updating parameter values, see Judd, Maliar, and Maliar (2011).

Moving forward, we will use the subroutines in this section to construct modified ADP-POST (MADP-POST) algorithms. In general, the purpose of this class of algorithms is improve the quality of the value functions and decision rules generated by the unmodified ADP-POST algorithm.

3 Common Global Solution Methods

In this section, I will briefly review common global solution methods for dynamic economic models, focusing primarily on those used most frequently to solve DSGE models. I will then

evaluate the performance of some of these methods in the following sections; and will compare the results to those of algorithms that make use of post-decision state variables. I will begin the overview with a review of value function iteration.

3.1 *Algorithm 2.1: Value Function Iteration*

Value function iteration (VFI) is used to solve infinite horizon dynamic programming problems. It involves discretizing the state space, iterating over all states, and then updating the value function approximation until some measure of convergence is achieved. It can be used as a standalone solution method or it can serve as a first step for more advanced solution methods. The basic algorithm for VFI is presented below. Since it can easily be extended to the heterogeneous agents case, I do not provide a separate solution algorithm.

The VFI algorithm proceeds as follows:

- *Step 0*: Discretize the state space. For a representative agent model, choose a grid for endogenous state variables, $\mathbf{s} = \{s_1, ..., s_N\}$, where $s_h < s_j$ if $h < j$. Use Tauchen's method (1986) to discretize the exogenous processes, $\mathbf{x}$, into a K-state Markov chain with an associated transition matrix, P, where element $p_{k,l}$ is the probability of transitioning from state k to l. For heterogeneous agents models, do this for both aggregate and individual-level state variables.

- *Step 1*: Initialize the value function, $V^0(s_i, x_j)$, at all grid points, $\mathbf{s}$ x $\mathbf{x}$.

- *Step 2*: For each grid point, (i,j), compute the value of choosing all possible post-decision values of $\mathbf{s}$, s_m: $w_m(s_i, x_j) = u(s_i, x_j, s_m) + \beta \sum_{l=1}^{K} p_{j,l} V^0(s_m, x_l)$.

- *Step 3*: For each grid point, (i,j), identify the index associated with the choice of endogenous state variables that maximizes $w_m(s_i, x_j)$, m^*. Construct V^1 as follows for all (i,j): $V^1(s_i, x_j) = w_{m^*}$.

- *Step 4*: Check for convergence using the following metric: $max_{i=1,...N, j=1,...,K} |V^1(s_i, x_j) - V^0(s_i, x_j)| \leq \epsilon$

If the convergence criterion is satisfied, stop the algorithm and compute the policy function. If not, set $V_0 = V_1$ and return to Step 1.

3.2 *Algorithm 2.2: Parameterized Expectations Approach (PEA)*

The Parameterized Expectations Approach (PEA) differs from the solution methods we have considered so far in that it does not involve the discretization of the state space. Instead, it

requires us to construct continuous approximations of the expectational components of the first order conditions. These approximations are created through the repetition of three steps. First, a set of approximating functions is chosen and parameterized. Second, decisions are simulated based on the chosen parameterization. And third, some convergence criterion is checked to determine if the simulated decisions were sufficiently close to optimal.

One important thing to note about PEA is that it is not vulnerable to the curse of dimensionality in the same sense that VFI and backwards recursion are. In solution methods that rely on iteration over discretized states, adding more state variables–and, in particular, continuous state variables–substantially increases the number of function evaluations that must be performed to identify the maximum. In contrast, with PEA, adding another state variable adds no such requirement. It does, however, make it more difficult to choose an initial parameterization for the expectational terms.

The algorithm below constructs a general description of PEA, following Heer and Maussner (2009) and Den Haan and Marcet (1990). While the algorithm is for the infinite horizon case, it can easily be extended to capture finite horizon applications.

- *Step 0*: Identify all model equations, including first order conditions, constraints, and exogenous processes.

- *Step 1*: Choose an initial functional form and parameterization for the expectational components of the first order conditions. Here, we will represent this by $G_h(s_{it}, x_{it}; \theta_h^0)$, where G_h is a function that approximates the h^{th} expectational term and using parameters θ_h^0. Typically, high order polynomials are used to construct G. Assume that the realization of the expectational component at time t+1 is ψ_{it+1}^h, which means that $\psi_{it+1}^h = G_h(s_{it}, x_{it}) + u_{it}$ and $u_{it} \sim iid$.

- *Step 2*: Using the expectational term approximations, the initial values, and T periods worth of simulated exogenous processes, compute the values of the endogenous, individual-level state variables in each period and for each agent. Increase the value of n, the parameter value iteration counter.

- *Step 3*: Discard observations for periods $\tilde{T}$ and lower to eliminate the bias introduced by initial values. Use the data for all periods after $\tilde{T}$, all agents, and all expectational components to estimate the parameter values using the following objective function: $\hat{\theta_n} = argmin_{\theta^n} \sum_{i \in I} \sum_{t=\tilde{T}+1}^{T-1} \sum_{h \in H} (\psi_{it+1}^h - G_h(s_{it}, x_{it}; \theta^n)^2$

- *Step 4*: Set $\theta^n = (1-\alpha_n)\theta^{n-1} + \alpha_n \hat{\theta^n}$. Check the following convergence criterion: $max(\hat{\theta^n} -$

$\theta^{n-1}) < \epsilon$, where max() is performed element-wise. If the condition is satisfied, then stop. Otherwise, return to Step 2.

It should be noted that the stochastic version of PEA–the one given in Algorithm 2.3–is most closely related to the ADP-POST family of algorithms. Both fall into category of stochastic simulation algorithms according to the taxonomy in Judd, Maliar, and Maliar (2011).

3.3 Algorithm 2.3: Projection Methods (PM)

Projection Methods (PM) are difficult to characterize because they cover a wide variety of approaches to solving multi-period choice problems. Broadly, PM consists of many tools that can be used to approximate common functions in choice problems (i.e. value functions, decision rules, expectational equations, etc.). The PM approach is typically performed by choosing the functional form for an approximation, computing the residual of that approximation, and then using some method to minimize the residual. Below, we will consider a basic algorithm for PM, which follows Heer and Maussner (2009) and Heer and Maussner (2008).

- *Step 0*: Identify a set of optimality conditions or functional equations, $\hat{g}(s, x; \theta)$, that have an associated residual. Choose a family of basis functions, $\xi(s, x)$, and a degree of approximation, ρ, where $\hat{g}(s, x; \theta) = \sum_{k=0}^{\rho} \phi_k \xi_k(s, x)$. Chebychev Polynomials are a common choice in the literature; and have the benefit of being an orthogonal class of polynomials, which reduces multicollinearity in Step 3.

- *Step 1*: Define the residual function $R(s, x; \theta) = G(\hat{g}(s, x; \theta))$.

- *Step 2*: Choose a projection function, χ_i, and a weighting function, w. Compute the inner product for all of the H optimality conditions or functional equations, where i=1,...,H: $\chi_i = \int_s \int_x w(s, x) R(s, x; \theta) \chi_i ds dx$

- *Step 3*: Choose the value of θ that yields $\chi_i = 0$ for i=1,...,H. Alternatively, find θ that minimizes $\int_s \int_x R(s, x; \theta)^2 ds dx$

- *Step 4*: Check the accuracy of the solution, $\hat{\theta}$. If the measure of error is too large, then increase ρ and return to Step 1. For heterogeneous agents models, nest Steps 1-4 within an aggregation algorithm.

For Step 0, consider the Bellman equation for an infinite horizon choice problem. We know that $V(s_i, x_j) = max_{s_m} u(s_i, x_j, s_m) + \beta \sum_{l=1}^{K} p_{j,l} V(s_m, x_l)$, which implies that $R(s_i, x_j; \theta) =$

$max_{S_m}[V(s_i, x_j) - u(s_i, x_j, s_m) - \beta \sum_{l=1}^{K} p_{j,l} V^0(s_m, x_l)]$. Here, an ideal choice of g() might be V(). However, we might think about choosing policy functions or Euler equations for g() instead.

In the next section, I will discuss the metrics that can be used to measure the accuracy of all of the aforementioned solution methods; and make it possible to make meaningful comparisons across alternatives.

4 Accuracy Measures

In many cases, it is possible to arbitrarily increase the accuracy of a solution method; however, doing so will typically incur a run time increase. For instance, with VFI, increasing the number of nodes in the state space grid will improve accuracy, but will also increase the amount of time it takes to perform an iteration. Similarly, with PEA and PM, accuracy can be improved by increasing the order of the polynomial transformations of state variables, but doing so may indirectly increase run time by necessitating a smaller stepsize. For this reason, it is often not meaningful to discuss "convergence time" and "accuracy" separately. Moving forward, we will structure the analysis around the tradeoff between convergence time and accuracy.

The primary measures of accuracy we explore are as follows: 1) Euler equation residuals; 2) dynamic Euler equation residuals; 3) deviations from the analytical decision rules; and 4) moments of cross-sectional distributions. Other popular measures of accuracy not considered include time series moment comparisons and the DM statistic (Den Haan and Marcet, 1994). Moment analysis of aggregate time series is omitted because it typically does not yield useful comparisons of global solution methods (Heer and Maussner, 2008). The DM statistic is dropped in favor of the dynamic Euler equation residual Den Haan (2010a), which fulfills the same role, but is arguably easier to interpret.

4.1 Euler Equation Residuals

Heer and Maussner (2009) find that there are substantial differences in accuracy between global and local solution methods when Euler equation residuals are used as the metric for accuracy. The general strategy behind Euler equation residual tests is to determine the accuracy of the decision rules by checking whether or not they are consistent with the model's optimality conditions. The Euler equation residual method is typically implemented by using some variation of the following five steps:

- *Step 1.* Identify the model's Euler equations and write them in terms of residual equations that are equal to zero if the agent has made an optimal decision.

- *Step 2.* Solve the model and recover the decision rules.
- *Step 3.* Plug the decision rules into the Euler equations.
- *Step 4.* Evaluate the residual equations over a discrete state space grid.
- *Step 5.* Compute descriptive statistics for the Euler equation residuals to use as a basis for comparison.

For the sake of consistency, I will construct Euler equation residuals in the style of Den Haan (2010a). In the simplest case, where capital is the only state and consumption is the only control, the Euler equation residual can be written as follows:

$$\left|\frac{c(k) - \bar{c}(k)}{\bar{c}(k)}\right| \tag{20}$$

In the equation above, c(k) is the consumption value implied by the decision rules and $\bar{c}(k)$ is the value computed explicitly from the conditional expectation (in stochastic models) on the right hand side of the Euler equation. The benefit of using this particular form of the Euler equation residual is that scale is not arbitrary or difficult to interpret. The residual is simply the percentage deviation of the consumption decision rule from optimality at a given node in the state space.[6] Note that residuals are typically computed at many points over the state space grid; and are then summarized by two statistics: the maximum and the average. It is also sometimes useful to identify the location of the maximum within the state space.

4.2 Dynamic Euler Equation Residuals

One limitation of the standard Euler equation residual is that it only captures "static inaccuracies" – that is, one-off-optimization errors. In order to get a more complete understanding of the accuracy of different solution methods, we must take an expanded view of errors that captures dynamics. In particular, it will be useful to measure the extent to which static inaccuracies accumulate over time in dynamic setting.

The DM statistic (Den Haan and Marcet, 1994) is an alternative to using Euler equation residuals that captures the accumulation of errors over time. It is difficult to interpret, however, since the frequency of rejection rises with the size of the simulation length. For this reason, I will consider the dynamic Euler equation residual, which fulfills the same role, but has a simpler interpretation. Following Den Haan (2010a), we may compute it as follows:

[6] For another form of the Euler equation residual with a clear interpretation, see Christiano and Fischer (2000).

- *Step 0.* Set $\tilde{k}_0 = k_0$.
- *Step 1.* Create a temporary variable: $\hat{k}_{t+1} = k(\tilde{k}_t, \epsilon_t)$.
- *Step 2.* Compute the conditional expectation of the right hand side of the Euler equation using $\hat{k}_{t+1}$, $k(\hat{k}_{t+1}, \epsilon_{t+1})$, and the distribution of ϵ_{t+1}.
- *Step 3.* Calculate $\bar{c}_t$ using the Euler equation, the budget constraint, and $\tilde{k}_{t+1}$.

We can now use equation (20) to compare c_t and $\bar{c}_t$. The important difference is that errors are now generated on the simulation path, rather than at each grid point in a predefined region within the state space. This accomplishes two things. First, it introduces dynamics into the error measure by allowing mistakes made in previous periods to carry over into future periods. And second, it only examines errors on the simulation path; and, therefore, eliminates the requirement that algorithms be tested in regions of the state space that are outside of the ergodic set.

5 Results

In this section, we will test the algorithms that were described earlier. This will proceed as follows. First, the baseline model will be presented. Next, the model will be solved using the aforementioned algorithms. Third, the results will be compared using the measures of accuracy described above. I will start by outlining a standard business cycle model with no labor supply, which will be used in the first set of tests. In the second set of tests, I will focus models with many heterogeneous agents.

5.1 Neoclassical Model (Infinite Horizon, Representative Agent)

Households in the baseline model are assumed to maximize expected, discounted utility from consumption, c_t:

$$max_{c_0} E_0 \sum_{t=0}^{\infty} \beta^t \left[log(c_t) \right] \tag{21}$$

It is assumed that $\beta \in (0, 1)$. Additionally, the household faces the following budget constraint, where k_t is the household's capital stock, and r_t is the return to capital:

$$k_{t+1} = (1 + r_t)k_t - c_t \tag{22}$$

The firm produces output using aggregate capital, K_t with the following production function:

$$Y_t = Z_t K_t^{\alpha} \tag{23}$$

Capital is assumed to depreciate at rate δ. Productivity, Z_t is assumed to follow an AR(1) process in logs:

$$lnZ_t = \rho lnZ_{t-1} + \epsilon_t, \;\; \epsilon_t \sim N(0, \sigma) \tag{24}$$

The firm maximizes profit, yielding the following factor price for capital:

$$r_t = \alpha Z_t K_t^{\alpha-1} - \delta \tag{25}$$

The economy is subject to an aggregate resource constraint:

$$Y_t = K_{t+1} - (1-\delta)K_t + C_t \tag{26}$$

Additionally, we assume that all individual variables are equal to aggregate variables in equilibrium: $c_t = C_t$ and $k_t = K_t$. Combining the first order conditions for the household, the equilibrium conditions, and the first order conditions for the firm, we get the following equations that characterize the model:

$$\frac{1}{c_t} = \lambda_t \tag{27}$$

$$\lambda_t = \beta E_t \lambda_{t+1} \left[\alpha Z_{t+1} K_{t+1}^{\alpha-1} + (1-\delta)\right] \tag{28}$$

$$K_{t+1} + C_t = Z_t K_t^{\alpha} + (1-\delta)K_t \tag{29}$$

Using the description above and the methods outlined earlier, it will now be possible to solve, simulate, and test the accuracy of the model under each of the different candidate solution methods. In particular, results will be presented for value function iteration, parameterized expectations, projection methods, and two versions of the ADP-POST algorithm.

5.2 Full Depreciation Case

In the case where $\delta = 1$, the model has a known, closed-form solution. In particular, the decision rule for capital is given as follows:[7]

$$k_{t+1} = \alpha\beta z_t k_t^{\alpha} \tag{30}$$

This special case provides a powerful tool for testing algorithms by making it possible to compare numerical solutions to the the exact, analytical solution. I take advantage of this feature by solving and simulating each algorithm for the neoclassical model with full depreciation. I then perform a visual comparison of the analytical and numerical decision rules. This allows me to identify where each algorithm's decision rules deviate from the model's analytical solution.

[7] See the appendix for a proof.

5.3 International Business Cycle Model (Infinite Horizon, Autarky)

The second set of tests will expand on the representative agent model by incorporating many heterogeneous countries. This is a simple extension of the original model, since all countries will be assumed to be ex-ante identical and will exist in autarky. Differences across countries will be generated by sequences of country-specific productivity shocks. The purpose of this exercise will be to test the performance of ADP-POST in a setting with heterogeneity. The results of this exercise can be interpreted as the amount of time it would take to compute the stationary distribution of wealth in an incomplete markets model, such as the one described by (Aiyagari, 1994). Alternatively, this can be interpreted as the time needed to complete a single iteration of the Krusell and Smith (1998) algorithm in a model with incomplete markets and aggregate uncertainty.

Moving forward, we will solve a substantially simpler model for three reasons. First and most importantly, doing so will limit the number of confounders. That is, the results will primarily reflect the time needed to solve the household's problem and simulate its behavior, rather than the behavior of supplementary algorithms that would be needed to solve larger models. Second, the simpler approach will make it possible to compare numerically-generated distributional moments to the analytically-derived distributional moments. And third, all of the solution methods tested–including the least stable and the slowest–will converge in a short window of time, allowing for a valid comparison across algorithms.

Below, we will consider the model in detail. Each country is assumed to consist of a representative agent who maximizes utility from consumption:

$$max_{c_{i0}} E_0 \sum_{t=0}^{\infty} \beta^t \left[log(c_{it}) \right] \tag{31}$$

Unlike the representative agent model, it will be assumed that the natural log of productivity is independently and identically distributed across time and across countries:

$$lnZ_{it} = \epsilon_{it}, \;\; \epsilon_{it} \sim N(0, \sigma) \tag{32}$$

The firm in each country produces output using aggregate capital, K_{it} with the following production function:

$$Y_{it} = Z_{it} K_{it}^{\alpha} \tag{33}$$

Each economy is subject to an aggregate resource constraint:

$$Y_{it} = K_{it+1} - (1 - \delta) K_{it} + C_{it} \tag{34}$$

Aggregate capital and consumption for the global economy are computed as follows, where μ_i is the mass of country i:

$$K_t = \sum_{i=1}^{N} K_{it}\mu_i \tag{35}$$

$$C_t = \sum_{i=1}^{N} C_{it}\mu_i \tag{36}$$

5.4 Representative Agent Model with Nonconvexities (Infinite Horizon)

Finally, we compare the stability of ADP-POST to PEA (the other stochastic simulation algorithm considered). This is done by introducing a non convexity into the household's decision rules along the capital dimension. I generate this by assuming that a minimum level of capital is needed to produce output. Otherwise, households engage in home production and receive $\bar{Y}$:

$$Y_t = \begin{cases} Z_t K_t^{\alpha}, & K > \tilde{K} \\ \bar{Y}, & K \leq \tilde{K} \end{cases} \tag{37}$$

This exercise will attempt to capture something that is elusive in the more formal accuracy and run time tests: solution methods also differ in terms of stability and ease of parameterization. In particular, ADP-POST approaches the stability of methods such as VFI and does not require a complicated parameterization. In contrast, other stochastic simulation methods will often diverge when nonconvexities are introduced if an extensive reparameterization is not performed.

5.5 Model Calibration

Both the representative agent model and the heterogeneous agents model were calibrated for an annual time period, using the parameter values given in Table 6. The annual time period allows me to use an autoregressive parameter for the productivity process that is further from unity.[8] Additionally, the time period provides an additional benefit when back-propagation is used in an infinite horizon model, as is described in Algorithm 3.3. in the appendix. In particular, it substantially reduces the computational intensity of solving the model.

[8]Many discrete approximations to continuous processes perform poorly when the series is highly persistent. In this case, I have used the Rouwenhorst (1995), which tends to perform better than other discretization methods when series are persistent, but is still subject to the same problem.

Parameter	Description	Value
β	Discount Factor	.95
α	Capital's Share in Production	.33
σ	Standard Deviation of Technology Shock	0.035
ρ	Technology Level AR(1) Coefficient	0.8
δ	Physical Capital Depreciation Rate	.1

Table 1: Baseline Parameter Values

5.6 Algorithm Refinements and Details

In earlier sections, I presented general algorithms for all of the solution methods that will be tested in the following section. However, for the purposes of this paper, I made several algorithmic refinements to ensure convergence in all of the tests. In particular, the algorithm for PEA was adjusted to incorporate dynamic bounds for the household-level capital grid. To understand how this works, consider a grid for capital, $k \in [k_1, ..., k_{kn}]$, which is centered around its steady state value, $k_{kn/2+1}$. When the algorithm is initiated, use the smaller grid, $k \in [k_{kn/2-j}, ..., k_{kn/2+j}]$, which is more tightly centered around the steady state value of capital. After each iteration, expand the grid by adding more nodes if any endpoint nodes were chosen in the previous step. Without this refinement, the performance of PEA was substantially worse; and divergence was more likely. For a full description of this method, see Maliar and Maliar (2005), which incorporates this subroutine to foster stability.

In addition to the PEA refinement, I also incorporated my own refinements for the ADP algorithms. In particular, for all ADP-POST and MADP-POST algorithms, I incorporated the stepsize algorithm described in Section 3.2.1. Additionally, for all MADP-POST algorithms, I included the refinement described in Section 3.2.2, which identifies and exploits the well-behaved region.

Finally, it is important to note that the PM approach used in this paper approximates the value function, rather than the Euler equations. The purpose behind this decision was to make the PM algorithm sufficiently distinct from the PEA algorithm, rather than performing the same set of tests twice. The algorithm works by first performing VFI on a small grid. The resulting look-up table approximation is then combined with the fixed point condition implied by the functional equations to perform the projection. In the heterogeneous agents model, both the PM and PEA algorithms use third order Chebychev polynomials of the first type to transform state variables in the approximation equations. Additionally, all continuous state

variable transformations are interacted with the agent's employment state. In the representative agent model, a simple, exponential polynomial of state variables is used to parameterize PEA.

5.7 Simulation Results

We will now examine the results by comparing the solution methods using run time, accuracy, and stability as criteria. Table 2 provides the run times for the version of the model with a closed form solution. Visual representations of the analytically-derived decision rules, numerical solution method-specific decision rules, and the error (the difference between the two) are given in Figures 3, 4, 5, and 6. Tables 3 and 4 show the results for the standard, representative agent business cycle model without labor. Table 5 shows the results for the international business cycle model for all solution methods other than ADP-POST; and Table 6 shows the results for the same model, but solved by MADP-IS-GPU on a grid with 200,000 nodes, rather than 10,000. All simulations were performed using a quad-core 3.40 gigahertz processor. Additionally, the MADP-IS-GPU algorithm performed all GPU operations on an NVIDIA GeForce GTX 560 Ti (2GB) with 384 CUDA cores.

METHOD	RT (nk=100)	RT (nk=1000)
VFI	00:00:21:43	00:33:39:70
PEA	00:00:00:37	00:00:00:40
PM	00:00:07:17	00:08:27:98
ADP	00:00:11:08	00:03:29:37
MADP	00:00:11:04	00:03:29.91

Table 2: Full Depreciation Business Cycle Model

Note that nk in Table 2 refers to the number of capital gridpoints used. "MAX(S)" and "MEAN(S)" in Tables 3 and 4 refer to the maximum and mean of the standard Euler equation residuals, which were computed at each state space node; whereas, "MAX(D)" and "MEAN(D)" refer to the dynamic Euler equation residuals, which were computed on a simulation path with identical draws for the exogenous processes. RT stands for run time and has the following format: hours:minutes:seconds:hundredths of a second. In the international business cycle model results, "STD(C)" and "STD(K)" stand for the standard deviation of consumption and the standard deviation capital, respectively.

Table 2 provides the results for the full depreciation business cycle model. Simulations in the first run time column were performed on a 300-node grid: 100 individual capital nodes and 3 productivity nodes. Here, VFI converged the slowest; and was followed by the ADP family

of algorithms, and then by projection methods. PEA was substantially faster than all other algorithms, converging in less than one second. Next, the state space was expanded in size to 3000 nodes (1000 individual capital nodes and 3 productivity nodes). This caused VFI's run time to increase by a factor of 94. Similarly, the run time for projection methods increased by a factor of 70, and was longer than the run time for the ADP algorithms. In contrast, the ADP algorithms were less affected by the curse of dimensionality and only experienced a run time increase by a factor of approximately 19. Since an increase in the number of capital nodes does not change the way in which the PEA algorithm is executed, there was no change in its run time; however, there were also no changes along other dimensions, such as accuracy.

Since the model used in this first set of tests has a closed form solution, we will analyze the accuracy of the results initially by comparing them to the analytical decision rules. The plots in column 1 of Figures 3 and 4 display the decision rules for capital that emerge from each of the solution methods. In each of the panels, the following period's capital stock is plotted against the current capital stock and productivity. A change in the level of productivity changes the relationship between k and k', which yields the three separate conditional decision rules shown in each panel. Based on the first column, VFI, PEA, PM, and MADP-POST appear to have decision rules that are visually similar to the analytical decision rules that emerge from the model's closed form solution.

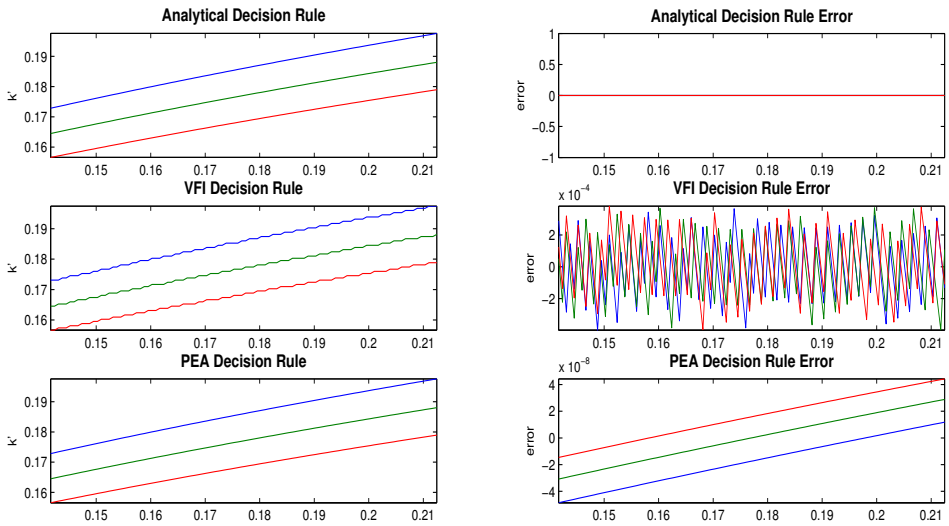

Figure 3: Decision Rules and Errors for AD, VFI, and PEA (100 Capital Nodes, 3 Productivity Nodes).

The second columns of Figures 3 and 4 display the difference between the solution method's implied k' and the k' that emerges from the analytical decision rules. From these plots, we can

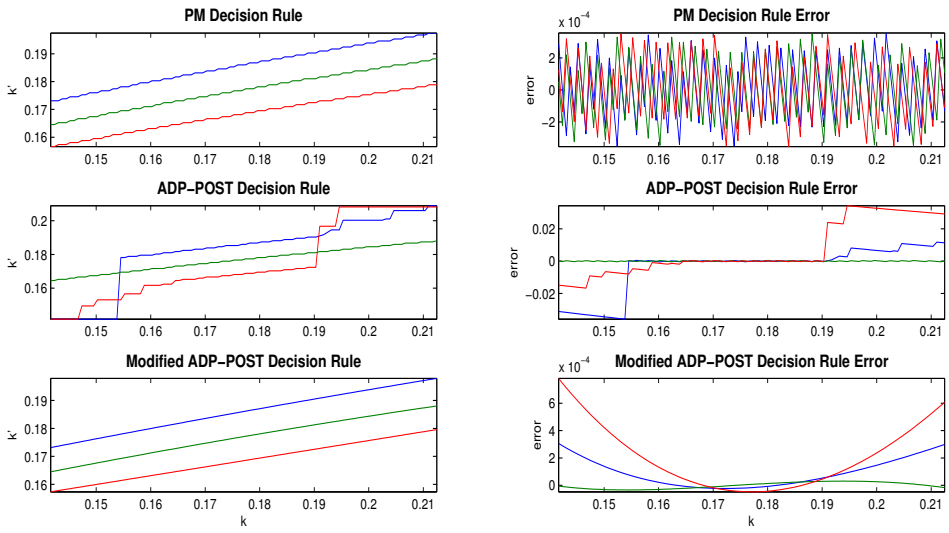

Figure 4: Decision Rules and Errors for PM, ADP-POST, Modified ADP-POST (100 Capital Nodes, 3 Productivity Nodes).

see that VFI, PEA, PM, and MADP-POST all differ only slightly from the analytical decision rules; however, the ways in which they differ are not identical across methods. For both VFI and PM, error sizes are almost uniformly distributed across the state space. In contrast, for PEA and MADP-POST–the stochastic simulation methods–errors are substantially smaller in the places most frequently visited on the simulation path. Unmodified ADP-POST yields very small errors within a subset of the ergodic set. However, it makes substantial errors in areas far away from the steady state, which are only infrequently visited on the simulation path.

Figures 5 and 6 provide the same information, but for the case with 1000 individual capital nodes and 3 productivity nodes. Comparing Figures 3 and 4 to 5 and 6, we can see that an increase in the number of individual capital nodes reduced the jaggedness of the decision rules for VFI, PM, ADP-POST, and MADP-POST; however, it did not change their general shapes. Additionally, with the respect to accuracy, all solution methods other than PEA appear to have improved by an order of magnitude with respect to their errors. In particular, the order of magnitude of the error for MADP-POST is now identical to that of VFI, even though MADP-POST converges in small fraction of the time it takes VFI to converge. Furthermore, the run time gap appears to be growing while the accuracy gap shrinks, suggesting that MADP-POST will be particularly useful for problems with very large state spaces.

Next, we will consider the other measures of performance. Table 3 provides the results for the standard business cycle model without labor. All simulations were performed on a 300-node grid: 100 individual capital nodes and 3 productivity nodes. As expected, VFI is

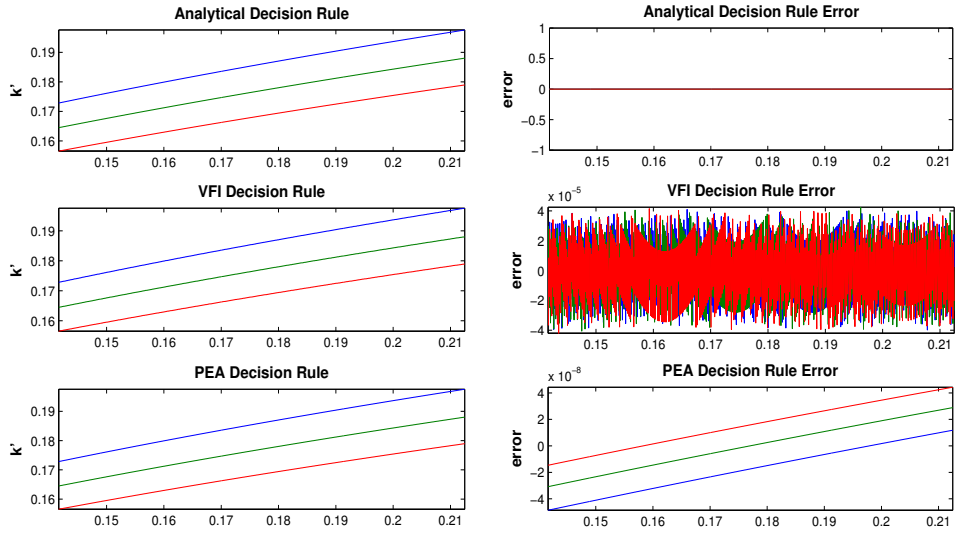

Figure 5: Decision Rules and Errors for AD, VFI, and PEA (1000 Capital Nodes, 3 Productivity Nodes)

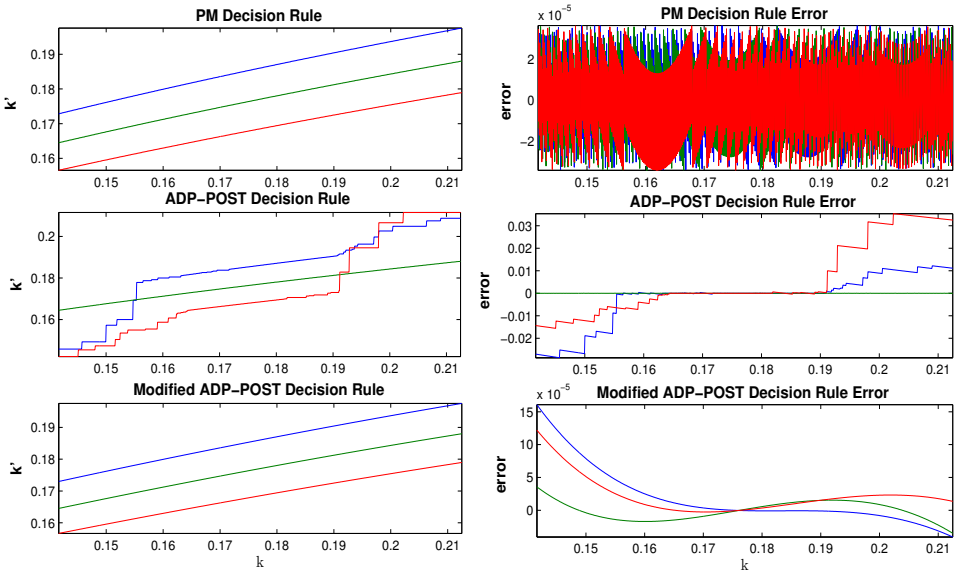

Figure 6: Decision Rules and Errors for PM, ADP-POST, and Modified ADP-POST (1000 Capital Nodes, 3 Productivity Nodes)

the most accurate solution method for this grid size as measured by the max standard Euler equation error, but is substantially slower than the alternatives. Additionally, VFI performs worse along other dimensions dimensions of accuracy, such as the mean standard Euler equation residual, and the mean and max dynamic Euler equation residual. In particular, MADP-POST appears to perform as well as or better than other algorithms with respect to measures of mean error. This is because MADP-POST achieves very low errors in places frequently visited on

the simulation path, but still retains a high degree of accuracy in areas infrequently visited, unlike the standard version of the ADP-POST algorithm. Not surprisingly, convergence times are similar to the full depreciation case. Here, VFI takes 3.10 times longer to converge than PM; and 2.05 times longer to converge than ADP. PEA is substantially faster than all other algorithms, converging in less than one second.

METHOD	MAX(S)	MEAN(S)	MAX(D)	MEAN(D)	RT
VFI	3.500E-3	1.500E-3	3.300E-3	1.300E-3	00:00:22:44
PEA	1.536E-1	5.280E-2	6.906E-4	3.585E-4	00:00:00:35
PM	3.700E-3	1.500E-3	3.300E-3	1.400E-3	00:00:07:23
ADP	2.847E-1	4.320E-2	3.300E-3	1.300E-3	00:00:10:94
MADP	6.900E-3	1.000E-3	9.555E-4	3.549E-4	00:00:10:94

Table 3: Standard Business Cycle Model (100 Individual Capital Nodes, 3 Productivity Nodes, 5000-Period Simulation Length)

Table 4 provides the results for the standard neoclassical model, but now with a 3000-node grid: 1000 individual capital nodes, and 3 productivity nodes. The purpose behind this simulation was to determine how relative performance changes as the size of the state space grows. As the table shows, all solution methods other than unmodified ADP-POST and PEA improved in accuracy by roughly one order of magnitude. The limited parameterization of PEA (a simple, exponential polynomial) fostered stability and rapid convergence, but did not allow for gains from increasing the size of the state space. Conversely, MADP-POST experienced substantial accuracy gains, and at a low cost: its run time increased by a factor of approximately 19. In contrast, PM took more 71 times longer to converge; and VFI took 88 times longer to converge. This suggests that MADP-POST will perform increasingly better than PM and VFI–in terms of the accuracy-run time tradeoff–as the state space grows. Furthermore, without an improved parameterization for PEA, it will be unable to generate the degree of accuracy obtainable with PM, VFI, and MADP-POST. It may also be prone to instability in more complex models.

Next, we'll consider the results for the international business cycle model in Tables 5 and 6. We will focus on comparing all solution methods to the version of the MADP-POST algorithm developed in this paper and described in Algorithm 1.2. Table 5 contains the simulations for everything other than Algorithm 1.2. All simulations were performed on a 10,000-node grid: 100 individual capital nodes and 100 productivity nodes. The results are displayed along three dimensions: the combined solution and simulation run time, the mean of the individual level

METHOD	MAX(S)	MEAN(S)	MAX(D)	MEAN(D)	RT
VFI	3.578E-4	1.494E-4	3.293E-4	1.475E-4	00:33:11:20
PEA	1.542E-1	5.240E-2	6.598E-5	3.453E-5	00:00:00:36
PM	3.912E-4	1.491E-4	3.595E-4	1.526E-4	00:08:34:85
ADP	2.210E-1	3.810E-2	4.500E-3	1.545E-4	00:03:29:15
MADP	1.400E-3	1.751E-4	1.876E-4	4.530E-5	00:03:28:35

Table 4: Full Depreciation Business Cycle Model (1000 Capital Nodes, 3 Productivity Nodes, 5000-Period Simulation Length)

variables (capital and consumption), and the standard deviation of the individual level variables. The method labelled "ADR" refers to results generated using the analytical decision rules.

METHOD	MEAN(C)	STD(C)	MEAN(K)	STD(K)	RT
ADR	0.3880	0.0144	0.1771	0.0067	—
VFI	0.3880	0.0156	0.1771	0.0069	22:38:58:85
PEA	0.3880	0.0144	0.1772	0.0066	00:01:46:58
PM	0.3880	0.0148	0.1771	0.0074	00:12:03:61

Table 5: International Business Cycle Model (100 Capital Nodes, 100 Productivity Nodes, 5000-Period Simulation Length)

Table 5 suggests that PEA largely outperforms VFI and PM. It takes fewer than 2 minutes to converge, and generates means and standard deviations that are close to those generated by iterating over the analytical decision rules with an identical sequence of shocks. The differences are particularly striking when compared to VFI, which takes more than 22 hours to converge and does not yield any improvements in accuracy.

METHOD	MEAN(C)	STD(C)	MEAN(K)	STD(K)	$\hat{\beta\alpha}$	RT
MADP-IS-GPU	0.3881	0.0144	0.1787	0.0066	0.3153	00:00:58:48

Table 6: International Business Cycle Model (1000 Capital Nodes, 200 Productivity Nodes, 5000-Period Simulation Length)

In Table 6, we consider a solution to the same model, but with *200,000* gridpoints: 1000 capital nodes and 200 productivity nodes. The only solution method we consider is MADP, augmented by the information-sharing algorithm (Algorithm 1.2) and GPU-computing. Here, we can see that MADP-IS-GPU is faster than all other algorithms listed in Table 5, including

PEA, even though the number of gridpoints has increased by a factor of 20. Furthermore, it provides highly accurate approximations of the cross-sectional standard deviations of capital and consumption. The means are also accurate, but are less accurate than those generated by other methods. Additionally a regression of period t+1 capital values on period t output yields a coefficient of 0.3153, which suggests that the numerical decision rule is very close to the analytical decision rule, which implies a coefficient of .3135 (i.e. $k_{t+1} = \alpha\beta y_t$).

It is important to note, however, that Table 6 is not directly comparable to Table 5. PEA, for instance, can be rewritten to exploit GPU-computing; and doing so would result in run time reductions for a given grid size. The purpose of this exercise was to demonstrate that an appropriately-parallelized version of the MADP-IS algorithm can exploit GPU-computing to generate massive convergence time reductions when at least one of two conditions is present: 1) the exogenous component of the state space is large; and 2) the post-decision state has a lower dimension than the pre-decision state. Note that we generate the second condition by assuming that z_t is i.i.d., rather than an AR(1) process, which allows us to represent the post-decision state by k_t. This does not change the dimensionality of the decision space, but does lower the dimension of the value function from $\bar{k}x\bar{z}$ to $\bar{k}$.

Next, we consider a test that compares the stability of PEA and ADP-POST. Thus far, all tests have been designed to ensure that each algorithm would converge with a parsimonious parameterization. However, methods such as PEA can become unstable when problems become more sophisticated. To demonstrate this, I modified the standard neoclassical model to generate a discontinuity in the decision rules, as described in Section 5.3. I then ran the programs for both PEA and ADP-POST without changing anything other than the model. The grids, parameterization, and starting values remained unchanged. PEA failed to capture the discontinuity and diverged. ADP-POST with no modifications converged. Figure 7 below shows the decision rule for consumption that emerged from ADP-POST.

This test described above is far from exhaustive, but demonstrates the usefulness of ADP-POST: it not only yields substantial convergence time reductions for a given level of accuracy, but is also able to capture any functional form–including one that incorporates nonconvexities–without requiring a reparameterization. This suggests that we can use ADP-POST to identify discontinuities within the ergodic set; and then apply a modified ADP-POST algorithm that explicitly accounts for the discontinuities identified.

Finally, we demonstrate the value of using the WLS modification of the ADP-POST algorithm described in Section 3.2.2. The purpose of this modification was to reduce the amount of weight placed on noisy observations in parts of the state space that were infrequently (or

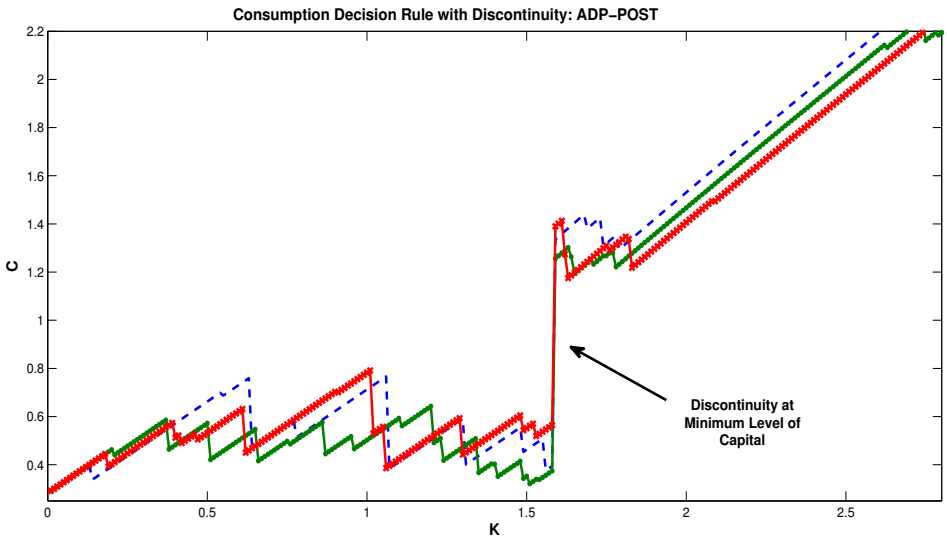

Figure 7: Consumption Decision Rule with Discontinuity (100 Capital Nodes, 3 Productivity Nodes, 5000-Period Simulation Length)

never) visited on the simulation path. This reduces a natural bias towards the initial values that is otherwise generated by ADP-POST algorithms. This test compares the coefficients from the modified ADP-POST algorithm under two estimation schemes: OLS and WLS with the weighting matrix described in Section 3.2.2. This comparison is done by performing the following regression:

$$k'(k,z) = \zeta z k^{\alpha} + \epsilon \tag{38}$$

Note that the analytical decision rule for the full depreciation model suggests that $\zeta = \beta\alpha$. Thus, we can use the estimate of ζ to check if the decision rules are accurate. Figure 8 plots $\hat{\zeta}$ for 50 iterations of the algorithm. For WLS, $\hat{\zeta} = \beta\alpha$ after 6 iterations. For OLS, this still has not occurred when the test ends after 50 iterations. This is because large parts of the state space are never explored, but continue to exert influence when all observations are given equal weight.

6 Conclusion

The ADP-POST algorithm provides a promising alternative to currently-used global solution methods for dynamic economic models. With the modifications introduced in this paper, it converges faster than VFI and PM, but typically slower than the stochastic version of PEA. Run time differences between the modified version of ADP-POST and VFI are exploding in the

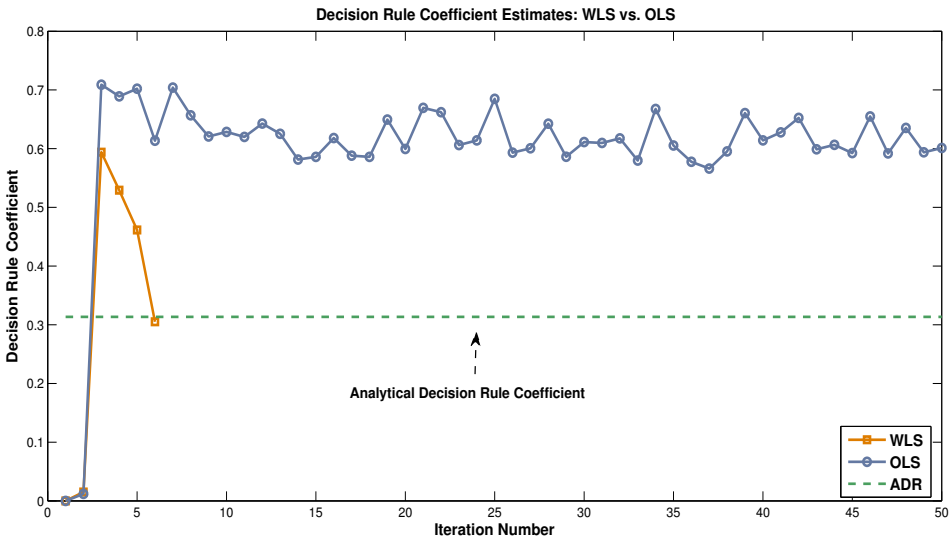

Figure 8: Decision Rule Coefficient Estimate in Full Depreciation Business Cycle Model (100 Capital Nodes, 100 Productivity Nodes, 5000-Period Simulation Length)

size of the state space; and become especially pronounced when the exogenous component of the state space is large, since ADP-POST does not require expectations to be computed.

In addition to achieving fast convergence, the modified version of ADP-POST yields Euler equation errors and dynamic Euler equation errors that are generally of the same order of magnitude as those generated by VFI. Additionally, the standard Euler equation errors from ADP-POST tend to be smaller than those generated by stochastic PEA in the tests performed. However, the unmodified version of ADP-POST generated results that were typically 1-2 orders of magnitude less accurate than the modified version introduced in this paper. Additionally, unmodified ADP-POST revealed a clear bias towards its initialization in tests that compared it to the analytical decision rules, but this was greatly mitigated by identifying and exploiting the well-behaved region in the modified version.

The only solution method that outperformed MADP-POST along both the run time and accuracy dimensions on certain tests was PEA. It is important to note, however, that PEA is both sensitive to parameterization and the choice of parameterized equations.[9] Furthermore, it can become increasingly difficult to parameterize PEA in large models; and accuracy gains beyond a certain magnitude will require the addition of higher order polynomial transformations of the state variables. In contrast, ADP-POST is capable of replicating the functional form of any decision rule, including decision rules that incorporate discontinuities. Additionally,

[9]Maliar and Maliar (2005) show that the choice of equation can lead to dramatic differences in convergence speed.

when combined with the information-sharing algorithm and GPU-computing in heterogeneous agent model applications, the algorithm introduced in this paper (MADP-IS-GPU) was able to converge faster and retain a high degree of accuracy, even though it was performed on a grid 20 times larger.

Finally, it is important to note that this paper does not fully showcase the set of tools that ADP-POST algorithms offer. In several places in the paper, I have mentioned that ADP-POST and MADP-POST are particularly well-suited to dealing with problems that have multiple, correlated exogenous processes or exogenous processes for which empirical data on exogenous processes is available. In the paper, I only provide algorithms for implementing these ideas– and not test results to demonstrate their advantages. In cases where exogenous processes are complicated and exhibit dependence, ADP-POST's value goes well beyond the gains it provides by reducing run times. It makes solving computational models more accessible; and permits economists to solve increasingly sophisticated models by simplifying the programming task. For this reason, formal test results for accuracy and run time may be less useful for these applications.

A Proof: Pre and Post-Decision Bellman Equation Relationships

In this subsection of the appendix, I will prove the relationships between the pre and post-decision Bellman equations that were stated in equations (10) and (12). I will start with (10), which makes the following claim:

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E[V(s_t, x_t)|s_t, x_{t-1}] \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} &= f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{39}$$

Now, recall equation (9), which states the value of being in post-decision state, (s_t, x_{t-1}):

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E\{max_{\{c_{t+s}\}_{s=0}^{\infty}} \sum_{s=0}^{\infty} \beta^s E[u(c_{t+s})|s_t, x_{t-1}]\} \\ \text{s.t. } s_{t+s+1} = T(c_{t+s}, s_{t+s}, x_{t+s}),\ c_{t+s} \in \Gamma(s_{t+s}, x_{t+s}),\ x_{t+s+1} &= f(x_{t+s}, \epsilon_{t+s}),\ \forall s = 0, 1, ... \end{aligned} \tag{40}$$

Notice that we may rewrite this equation using the Law of Iterated Expectations (LIE):

$$V^x(s_t, x_{t-1}) = E\{max_{\{c_{t+s}\}_{s=0}^{\infty}} \sum_{s=0}^{\infty} \beta^s E[u(c_{t+s})|s_t, x_t]|s_t, x_{t-1}\}$$
$$\text{s.t. } s_{t+s+1} = T(c_{t+s}, s_{t+s}, x_{t+s}),\ c_{t+s} \in \Gamma(s_{t+s}, x_{t+s}),\ x_{t+s+1} = f(x_{t+s}, \epsilon_{t+s}),\ \forall s = 0, 1, ... \tag{41}$$

Next, use the definition of the value of state (s_t, x_t):

$$V(s_t, x_t) = max_{\{c_{t+s}\}_{s=0}^{\infty}} \sum_{s=0}^{\infty} \beta^s E[u(c_{t+s})|s_t, x_t] \tag{42}$$

Substituting (A.4) into (A.3) yields (A.1), which was our original claim:

$$\begin{aligned} V^x(s_t, x_{t-1}) &= E[V(s_t, x_t)|s_t, x_{t-1}] \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} &= f(x_t, \epsilon_{t+1}),\ \forall t = 0, 1, ... \end{aligned} \tag{43}$$

We will now prove the claim from equation (12) that demonstrates the connection between the pre-decision and post-decision Bellman equations:

$$V(s_t, x_t) = max_{c_t} u(c_t) + \beta V^x(s_{t+1}, x_t) \tag{44}$$

Start by recalling the standard Bellman equation for a stochastic dynamic programming problem:

$$\begin{aligned} V(s_t, x_t) &= max_{c_t} u(c_t) + \beta E[V(s_{t+1}, x_{t+1})|s_t, x_t] \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} &\in f(x_t, \epsilon_{t+1}) \end{aligned} \tag{45}$$

Next, note that (s_t, x_t)–the pre-decision state–pins down c_t, which then pins down s_{t+1}:

$$s_{t+1} = T(\Gamma(s_t, x_t), s_t, x_t) \tag{46}$$

This suggests that the information set can be written as (s_{t+1}, x_t):

$$\begin{aligned} V(s_t, x_t) &= max_{c_t} u(c_t) + \beta E[V(s_{t+1}, x_{t+1})|s_{t+1}, x_t] \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} &\in f(x_t, \epsilon_{t+1}) \end{aligned} \tag{47}$$

Recalling the relationship derived earlier in equation (A.5), we may rewrite this to yield (12), which is what we wanted to prove:

$$\begin{aligned} V(s_t, x_t) &= max_{c_t} u(c_t) + \beta V^x(s_{t+1}, x_t) \\ \text{s.t. } s_{t+1} = T(c_t, s_t, x_t),\ c_t \in \Gamma(s_t, x_t),\ x_{t+1} &\in f(x_t, \epsilon_{t+1}) \end{aligned} \tag{48}$$

B Proof: Closed-Form Solution for Full Depreciation Case

Assume that z is a Markov process. Additionally, let next period variables be denoted with primes. That is, $x_{t+1} = x'$. Finally, assume that $\delta = 1$ and recall the budget constraint and Euler equation for capital from the representative agent model described earlier:

$$c = zk^{\alpha} - k' \tag{49}$$

$$\frac{1}{c} = \beta E\left\{\frac{\alpha z'}{c'}\middle| z\right\} \tag{50}$$

Now, assume that the decision rule for capital takes the following form:

$$\Xi(k, z) = g_0 + g_1 z k^{\alpha} \tag{51}$$

Next, plug the budget constraint into the Euler equation for capital:

$$\frac{1}{zk^{\alpha} - k'} = \beta E\left\{\frac{\alpha z' k'^{\alpha-1}}{z'k'^{\alpha} - k''}\middle| z\right\} \tag{52}$$

Now, add the assumed decision rule:

$$\frac{1}{(1-g_1)zk^{\alpha} - g_0} = \beta E\left\{\frac{\alpha z'(g_0 + g_1 z k^{\alpha})^{\alpha-1}}{z'(g_0 + g_1 z k^{\alpha})^{\alpha} - g_0 - g_1 z'(g_0 + g_1 z k^{\alpha})^{\alpha}}\middle| z\right\} \tag{53}$$

To simplify things, we will look for a solution where $g_0 = 0$.[10] This allows us to simplify the above equation as follows:

$$\frac{1}{(1-g_1)zk^{\alpha}} = \beta E\left\{\frac{\alpha z'(g_1 z k^{\alpha})^{\alpha-1}}{z'(g_1 z k^{\alpha})^{\alpha} - g_1 z'(g_1 z k^{\alpha})^{\alpha}}\middle| z\right\} \tag{54}$$

$$\rightarrow \frac{1}{(1-g_1)zk^{\alpha}} = \beta E\left\{\frac{\alpha}{(1-g_1)g_1 z k^{\alpha}}\middle| z\right\} \tag{55}$$

$$\rightarrow g_1 = \alpha\beta \tag{56}$$

We can then substitute this into our initial guess for the decision rule, yielding the following:

$$k' = \alpha\beta z k^{\alpha} \tag{57}$$

Note that we have demonstrated that the decision rule given above is compatible with the model's first order conditions.

[10] It can be shown that there will be no solution for the case when $g_0 \neq 0$.

C Finite Horizon Algorithms and Extensions

The primary focus of this research is to test and extend the infinite-horizon, ADP-POST framework as a solution method for dynamic economic models. However, several modified versions of this framework can be used to solve finite horizon models; and may provide useful tools for OLG macro models with many heterogeneous agents. For this reason, I've included a brief appendix section with an overview of such methods. The proceeding subsections are ordered as follows. First, I provide a basic algorithm for solving finite horizon problems using pre-decision state dynamic programming (backwards recursion). Next, I present the "double pass" version of finite horizon dynamic programming using ADP-POST. This includes a subroutine called back-propagation. Finally, I provide a brief explanation of how this "double pass" strategy might be employed to solve infinite horizon problems as well.

C.1 Algorithm 3.1: Backwards Recursion

The basic algorithm for backwards recursion proceeds as follows:

- *Step 0*: Discretize the state space. Choose a grid for endogenous, aggregate state variables, $\mathbf{s} = \{s_1, ..., s_N\}$, where $s_h < s_j$ if $h < j$. Discretize the exogenous processes, x, into a K-state Markov chain with an associated transition matrix, P, where element $p_{k,l}$ is the probability of transitioning from state k to l. For heterogeneous agents models, do this for both aggregate and individual-level state variables. Choose a maximum age, $\bar{a}$, where $\mathbf{A} = 1, ..., \bar{a}$.

- *Step 1*: At $\bar{a}$, solve $V_{\bar{a}}(s_i, x_j) = max_{s_m} u(s_i, x_j, s_m)$, at all grid points, $\mathbf{s}$ x $\mathbf{x}$.

- *Step 2*: Using the value of each state at age $\bar{a}$ determined in Step 1, compute the value of choosing all possible post-decision state, s_m, at age $\bar{a} - 1$: $w_m^{\bar{a}-1}(s_i, x_j) = u(s_i, x_j, s_m) + \beta \sum_{l=1}^{K} p_{j,l} V^{\bar{a}}(s_m, x_l)$.

- *Step 3*: For each grid point, (i,j), identify the index associated with the choice of endogenous state variables that maximizes $w_m(s_i, s_j)$, m^*. Construct $V^{\bar{a}-1}$ as follows for all (i,j): $V^{a_{max-1}}(S_i, X_j) = w_{m^*}$.

- *Step 4*: Repeat Steps 2-3 for $\bar{a} - 2, ..., 1$. Use the resulting age-specific value functions to determine the age-specific policy functions. For heterogeneous agents models, nest Steps 1-4 within an aggregation algorithm, as described earlier.

Since VFI and backwards recursion both revolve around discretizing the state space, they also both suffer from the curse of dimensionality when the household's choice problem involves

many continuous state variables. Similarly, improvements to both approaches rely on the same family of techniques. In particular, linear and cubic interpolation are two common strategies for overcoming the curse of dimensionality when such solution methods are applied.

C.2 Algorithm 3.2 Double-Pass ADP-POST (OLG)

The algorithm below is referred to as a "double-pass" version of ADP-POST because it contains both the forward simulation component from the finite horizon analog of the ADP-POST algorithm and a "back-propagation" step, described below.

The Double-Pass ADP-POST (OLG) Algorithm proceeds as follows:

- *Step 0*: i. Perform all initializations.
  - Initialize the "look-up table" approximation of the value function in all states. That is, for each state, $(s \in \mathbf{s}, x \in \mathbf{x}, a \in \mathbf{a}, S \in \mathbf{S}, X \in \mathbf{X})$, assign a starting value to $\overline{V}_a^0(S, X, s, x)$, where the superscript 0 indicates that this is an approximation taken at the 0th iteration and where a is the household's age.[11] Note that capital letters represent aggregate variables.
  - ii. Initialize the state, $(s_0 \in \mathbf{s}, x_0 \in \mathbf{x}, a_0 \in \mathbf{a}, S_0 \in \mathbf{S}, X_0 \in \mathbf{X})$. For overlapping generations models, there may be natural initial states for idiosyncratic shocks and endogenous, individual-specific state variables (i.e. a household may start with no assets), but aggregate-level state variables will depend on period when the household enters the model–and, thus, must be initialized separately for each agent.[12]
  - iii. Initialize the aggregation method states using the Krusell and Smith (1998) algorithm or the explicit aggregation method (Den Haan and Rendahl, 2010).
  - iv. Increment the iteration counter to n=n+1.
- *Step 1*: Choose a simulation period length, T. Generate a sample of the exogenous processes, $(X_t, x_{a,t})$, for the simulation period (1,...,T) and for all ages $(1, ...\bar{a})$, where $x_{a,t}$ denotes the set of individual-specific state variables for the age a household at time t.
- *Step 2*: For all periods, t=1,...T, and all ages, 1,...,$\bar{a}$, perform the following four steps:

[11] Recall that finite horizon dynamic programming problems require us to use different approximations of the value function at each age, since the structure of the choice problem fundamentally changes over the lifecycle. Additionally, note that we could also include $a \in \mathbf{a}$ as a deterministic state variable.

[12] For a more complete treatment of initializing state variables in OLG macro models, see Heer and Maussner (2009).

– i. Perform the "forward pass" portion of the algorithm. Choose controls to maximize the expression inside of the expectation of $V_a(S_{t-1}, X_{t-1}, s_{a,t}, x_{a-1,t-1})$ for a particular realization of $(X_t, x_{a,t})$:
$\tilde{V}_a^n(S_{t-1}, X_{t-1}, s_{a,t}, x_{a-1,t-1}) = max_{C_t} u(C_t) + \beta \overline{V}_{a+1}^{n-1}(S_t, X_t, s_{a+1,t+1}, x_{a,t})$

– ii. Perform back-propagation for each agent. This involves updating V_a^n using information gained about the values of states from future periods, but on the same state trajectory. This can be accomplished by recursively computing $\tilde{V}_a$ in each period each agent is alive, starting in their respective terminal periods:
$\tilde{V}_a^n(S_{t-1}, X_{t-1}, s_{a-1,t-1}, x_{a-1,t-1}) = u(\hat{C}_t) + \beta \tilde{V}_{a+1}^{n-1}(S_t, X_t, s_{a+\hat{1},t+1}, x_{a,t})$
Note that carets are used to denote endogenous state variables that were selected on the forward-pass step of the algorithm.

– iii. Compute the expectation by updating the value function approximation, V_a^{n-1}:
$\overline{V_a^n}(S_{t-1}, X_{t-1}, s_{a,t}, x_{a-1,t-1}) = (1 - \alpha_{n-1}) \bar{V}_a^{n-1}(S_{t-1}, X_{t-1}, s_{a,t}, x_{a-1,t-1})$
$+ \alpha_{n-1} \tilde{V}_a^n(S_{t-1}, X_{t-1}, s_{a,t}, x_{a-1,t-1})$

– iv. Use the result from the maximization step, $s_{a+1,t+1}$, the realization of the exogenous processes, $(X_{t+1}, x_{a+1,t+1})$, and the aggregate state, S_{t+1}, to compute the next period's pre-decision state.

- *Step 3:* Check convergence criterion. If satisfied, go to Step 4. If not satisfied, increment n and go to Step 1.

- *Step 4:* Update the paths and laws of motion for the aggregate states. Check the aggregate convergence criterion. If satisfied, go to Step 5. If not satisfied, go to Step 1.

- *Step 5:* Return the value function approximation, $V_a(S, X)^N$, in the form of a look-up table.

The most important departure from ADP-POST can be found in Step 2(ii). Here, we perform the back-propagation step, which permits us to construct an unbiased approximation of the value function. When the "forward pass" portion of the algorithm is performed, it generates a value function approximation that depends on two things: 1) utility function evaluations on the simulation path; and 2) post-decision state value initializations. However, when $\tilde{V}_{a+1}^n$ replaces $\bar{V}_{a+1}^{n-1}$ as an update to $\tilde{V}_a^n$, the value function approximation incorporates information about the utility derived in future states on this particular simulation trajectory.

Finally, we will consider an extension of the back-propagation mechanism to the infinite horizon case.

C.3 Algorithm 3.3: Double-Pass ADP-POST (Infinite Horizon)

Algorithm 3.3 differs from Algorithm 3.2 in two important ways. First, it covers the infinite horizon case. And second, it only performs the back-propagation step on a subset of the simulated data: $1,..,\bar{T}$, where $T > \bar{T}$. The reason why only a subset of the data is used is because $\tilde{\bar{V}}^n$ is recursively constructed, which means that it will be downward-biased in finite sample approximations, since it will miss the value attributable to periods $T+1, ...\infty$. Now, if an agent is in period 10 and T is large, then the recursively computed value function approximation in that period will not be substantially downward-biased, since T-10 is large, which means that any value from $T+1, ..., \infty$ will be heavily discounted and therefore negligible. On the other hand, if the agent is at T-1, then the future streams of utility from $T+1, ..., \infty$ will be important, which will yield a substantial downward-bias. For this reason, the algorithm only performs the back-propagation step on $1, ..., \bar{T}$. Furthermore, note that the larger $T - \bar{T}$ is, the less biased the value function approximations will be, but the longer it will take to achieve convergence, since it will not be possible to fully exploit back-propagation.

- *Step 0*: See Algorithm 3.2.
- *Step 1*: Choose a simulation period length, T, and a cutoff length, $\bar{T}$, where $T > \bar{T}$. Generate a sample of the exogenous processes, X_t, for the simulation period: 1,...,T.
- *Step 2*: For periods, t=1,...,T, perform the following five steps:
  - i. Choose controls to maximize the expression inside of the expectation of $V(S_{t-1}, X_{t-1})$ for a particular realization of X_t: $\tilde{V}^n(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) = max_{c_t} u(c_t) + \beta \overline{V}^{n-1}(S_t, X_t, s_{it+1}, x_{it})$
  - ii. Perform the back-propagation step. This involves updating V^n using information gained about the values of states from future periods, but on the same state trajectory. This can be accomplished by recursively computing $\tilde{\bar{V}}^n$ in each period, starting in period T: $\tilde{\bar{V}}^n(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) = u(\hat{c_t}) + \beta \tilde{V}^{n-1}(S_t, X_t, \hat{s_{it+1}}, x_{it})$ Note that carets are used to denote household-level endogenous state variables that were selected on the forward-pass step of the algorithm.
  - iii. For 1,...,$\bar{T}$, set $\tilde{V}^n(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) = \tilde{\bar{V}}^n(S_{t-1}, X_{t-1}, s_{it}, x_{it-1})$.
  - iv. Compute the expectation by updating the value function approximation, V^{n-1}: $\overline{V^n}(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) = (1 - \alpha_{n-1})\bar{V}^{n-1}(S_{t-1}, X_{t-1}, s_{it}, x_{it-1}) + \alpha_{n-1}\tilde{V}^n(S_{t-1}, X_{t-1}, s_{it}, x_{it-1})$
  - v. Use the result from the maximization step, s_{it+1}, the realization of the exogenous processes, X_{t+1}, x_{it+1}, and the aggregate state, S_{t+1}, to compute the next period's

pre-decision state.

- *Step 3:* Check convergence criterion. If satisfied, go to Step 4. If not satisfied, increment n and go to Step 1.
- *Step 4:* Return the value function approximation, $V(s,x)^N$, in the form of a look-up table.

Finally, note that the information-sharing subroutine in Algorithm 1.2 can be applied to Algorithms 3.2 and 3.3 to improve performance.

References

Aiyagari, S.R. 1994. "Uninsured Idiosyncratic Risk and Aggregate Saving." *Quarterly Journal of Economics* Vol. 109, 659-684.

Bellman, R. 1954. "The Theory of Dynamic Programming." *Bulletin of the American Mathematical Society* Vol 60, 503515.

Bertsekas, D. 2011. "Chapter 6: Approximate Dynamic Programming." In *Dynamic Programming and Optimal Control*. Vol. II, 3rd Edition. MIT. Draft.

Sutton, R., & Barto, A. 1998. *Reinforcement Learning*. Cambridge, Massachusetts: The MIT Press.

Christiano, L. & Fisher, J. 2000. "Algorithms for Solving Dynamic Models with Occasionally Binding Constraints." *Journal of Economic Dynamics and Control* Vol. 24, 1179-1232.

Darken, C. & Moody, J. 1992. "Towards Faster Stochastic Gradient Search," in J. Moody, D.L. Hanson, & R.P. Lippmann, eds, *Advances in Neural Information Processing Systems 4*, 1009-1016.

DeJong, D., & Dave, C. 2007. *Structural Macroeconometrics*. Princeton, New Jersey: Princeton University Press.

Den Haan, W.J. 2010. "Comparison of Solutions to the Incomplete Markets Model with Aggregate Uncertainty." *Journal of Economic Dynamics and Control* Vol. 34, 4-27.

Den Haan, W.J. 2010. "Assessing the Aggregate Law of Motion in Models with Heterogeneous Agents." *Journal of Economic Dynamics and Control* Vol. 34, 79-99.

Den Haan, W.J. & Marcet, A. 1994. "Accuracy in Simulations." *Review of Economic Statistics* Vol. 8, 31-34.

Den Haan, W.J. & Marcet, A. 1990. "Solving the Stochastic Growth Model by Parameterizing Expectations." *Journal of Business and Economic Statistics*, Vol. 8, 31-34.

Den Haan, W.J. & Rendahl, P. 2010. "Solving the Incomplete Markets Model with Aggregate Uncertainty Using Explicit Aggregation." *Journal of Economic Dynamics and Control* Vol. 34(1), 69-78.

Duffy, J. & McNelis, P. 2001. "Approximating and Simulating the Stochastic Growth Model: Parameterized Expectations, Neural Networks, and the Genetic Algorithm." *Journal of Economic Dynamics and Control* Vol. 25, 1273-1303.

Heer, B., & Maussner, A. 2008. "Computation of Business Cycle Models: A Comparison of Numerical Methods." *Macroeconomic Dynamics* Vol. 12, 641-663.

Heer, B., & Maussner, A. 2009. *Dynamic General Equilibrium Modeling: Computational Methods and Applications*. 2nd Edition. Berlin: Springer.

Judd, K. 1998. *Numerical Methods in Economics*. Cambridge, MA, London: MIT Press.

Judd, K. 1992. "Projection Methods for Solving Aggregate Growth Models." *Journal of Economic Theory* Vol. 58(2), 410-452.

Judd, K., Maliar, L., & Maliar, S. 2011. "Numerically Stable and Accurate Stochastic Simulation Approaches for Solving Dynamic Economic Models." *Quantitative Economics* Vol. 2(2), 173-210.

Krusell, P. & Smith, A. 1998. "Income and Wealth Heterogeneity in the Macroeconomy." *Journal of Political Economy* Vol. 106, 867-96.

Marcet, A. 1988. "Solving Nonlinear Models by Parameterizing Expectations." Working Paper, Carnegie-Mellon University.

McGrattan, E. 1999. "Application of Weighted Residual Methods to Dynamic Economic Models." In Ramon Marimon and Andrew Scott, *Computational Methods for the Study of Dynamic Economies*. Oxford and New York: Oxford University Press, 114-142.

Maliar, L. & Maliar, S. 2005. "Parameterized Expectations Algorithm: How to Solve for Labor Easily." *Computational Economics, Society for Computational Economics* Vol. 25(3), 269-274.

Maliar, L., Maliar, S., & Valli, F. 2010. "Solving the Incomplete Markets Model with Aggregate Uncertainty Using the Krusell-Smith Algorithm." *Journal of Economic Dynamics and Control* Vol. 34(1), 42-49.

Maxwell, M. 2011. "Approximate Dynamic Programming Policies and Performance Bounds for Ambulance Redeployment." Dissertation, Graduate School of Cornell University.

Powell, W. 2007. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, New Jersey.

Powell, W. 2012. "Perspectives of Approximate Dynamic Programming," *Annals of Operations Research*, Springer, DOI 10.1007/s10479-012-1077-6.

Powell, W., & Ryzhov, I. 2010. "Approximate Dynamic Programming With Correlated Bayesian Beliefs," *Proceedings of the 48th Allerton Conference on Communication, Control, and Computing.*

Rouwenhorst, K.G. 1995. "Asset Pricing Implications of Equilibrium Business Cycle Models." In: Cooley, T.F. (Ed.), *Frontiers of Business Cycle Research.* Princeton, NJ: Princeton University Press.

Sargent, T.J. 1987. *Dynamic Macroeconomic Theory.* Cambridge, MA: Harvard University Press.

Simao, H., Day, J., Abraham, G., Gifford, T., Nienow, T., and Powell, W. (2009). An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application. *Transportation Science* Vol. 43(2), 178-197.

Tauchen, G. 1986. "Finite State Markov-Chain Approximations to Univariate and Vector Autoregressions." *Economic letters* Vol. 20, 177-181.

Van Roy, B., Bertsekas, D., Lee, Y., & Tsitsiklis, J.N. 1997. A Neurodynamic Programming Approach to Retailer Inventory Management. In *Proceedings of the 36th IEEE Conference on Decision and Control* Vol. 4, 4052-4057.

Earlier Working Papers:

For a complete list of Working Papers published by Sveriges Riksbank, see www.riksbank.se

Estimation of an Adaptive Stock Market Model with Heterogeneous Agents *by Henrik Amilon*	2005:177
Some Further Evidence on Interest-Rate Smoothing: The Role of Measurement Errors in the Output Gap *by Mikael Apel and Per Jansson*	2005:178
Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through by Malin Adolfson, Stefan Laséen, Jesper Lindé and Mattias Villani	2005:179
Are Constant Interest Rate Forecasts Modest Interventions? Evidence from an Estimated Open Economy DSGE Model of the Euro Area *by Malin Adolfson, Stefan Laséen, Jesper Lindé and Mattias Villani*	2005:180
Inference in Vector Autoregressive Models with an Informative Prior on the Steady State *by Mattias Villani*	2005:181
Bank Mergers, Competition and Liquidity *by Elena Carletti, Philipp Hartmann and Giancarlo Spagnolo*	2005:182
Testing Near-Rationality using Detailed Survey Data *by Michael F. Bryan and Stefan Palmqvist*	2005:183
Exploring Interactions between Real Activity and the Financial Stance *by Tor Jacobson, Jesper Lindé and Kasper Roszbach*	2005:184
Two-Sided Network Effects, Bank Interchange Fees, and the Allocation of Fixed Costs *by Mats A. Bergman*	2005:185
Trade Deficits in the Baltic States: How Long Will the Party Last? *by Rudolfs Bems and Kristian Jönsson*	2005:186
Real Exchange Rate and Consumption Fluctuations follwing Trade Liberalization *by Kristian Jönsson*	2005:187
Modern Forecasting Models in Action: Improving Macroeconomic Analyses at Central Banks *by Malin Adolfson, Michael K. Andersson, Jesper Lindé, Mattias Villani and Anders Vredin*	2005:188
Bayesian Inference of General Linear Restrictions on the Cointegration Space *by Mattias Villani*	2005:189
Forecasting Performance of an Open Economy Dynamic Stochastic General Equilibrium Model *by Malin Adolfson, Stefan Laséen, Jesper Lindé and Mattias Villani*	2005:190
Forecast Combination and Model Averaging using Predictive Measures *by Jana Eklund and Sune Karlsson*	2005:191
Swedish Intervention and the Krona Float, 1993-2002 *by Owen F. Humpage and Javiera Ragnartz*	2006:192
A Simultaneous Model of the Swedish Krona, the US Dollar and the Euro *by Hans Lindblad and Peter Sellin*	2006:193
Testing Theories of Job Creation: Does Supply Create Its Own Demand? *by Mikael Carlsson, Stefan Eriksson and Nils Gottfries*	2006:194
Down or Out: Assessing The Welfare Costs of Household Investment Mistakes *by Laurent E. Calvet, John Y. Campbell and Paolo Sodini*	2006:195
Efficient Bayesian Inference for Multiple Change-Point and Mixture Innovation Models *by Paolo Giordani and Robert Kohn*	2006:196
Derivation and Estimation of a New Keynesian Phillips Curve in a Small Open Economy *by Karolina Holmberg*	2006:197
Technology Shocks and the Labour-Input Response: Evidence from Firm-Level Data *by Mikael Carlsson and Jon Smedsaas*	2006:198
Monetary Policy and Staggered Wage Bargaining when Prices are Sticky *by Mikael Carlsson and Andreas Westermark*	2006:199
The Swedish External Position and the Krona *by Philip R. Lane*	2006:200

Price Setting Transactions and the Role of Denominating Currency in FX Markets *by Richard Friberg and Fredrik Wilander*	2007:201
The geography of asset holdings: Evidence from Sweden *by Nicolas Coeurdacier and Philippe Martin*	2007:202
Evaluating An Estimated New Keynesian Small Open Economy Model *by Malin Adolfson, Stefan Laséen, Jesper Lindé and Mattias Villani*	2007:203
The Use of Cash and the Size of the Shadow Economy in Sweden *by Gabriela Guibourg and Björn Segendorf*	2007:204
Bank supervision Russian style: Evidence of conflicts between micro- and macro-prudential concerns *by Sophie Claeys and Koen Schoors*	2007:205
Optimal Monetary Policy under Downward Nominal Wage Rigidity *by Mikael Carlsson and Andreas Westermark*	2007:206
Financial Structure, Managerial Compensation and Monitoring *by Vittoria Cerasi and Sonja Daltung*	2007:207
Financial Frictions, Investment and Tobin's q *by Guido Lorenzoni and Karl Walentin*	2007:208
Sticky Information vs Sticky Prices: A Horse Race in a DSGE Framework *by Mathias Trabandt*	2007:209
Acquisition versus greenfield: The impact of the mode of foreign bank entry on information and bank lending rates *by Sophie Claeys and Christa Hainz*	2007:210
Nonparametric Regression Density Estimation Using Smoothly Varying Normal Mixtures *by Mattias Villani, Robert Kohn and Paolo Giordani*	2007:211
The Costs of Paying – Private and Social Costs of Cash and Card *by Mats Bergman, Gabriella Guibourg and Björn Segendorf*	2007:212
Using a New Open Economy Macroeconomics model to make real nominal exchange rate forecasts *by Peter Sellin*	2007:213
Introducing Financial Frictions and Unemployment into a Small Open Economy Model *by Lawrence J. Christiano, Mathias Trabandt and Karl Walentin*	2007:214
Earnings Inequality and the Equity Premium *by Karl Walentin*	2007:215
Bayesian forecast combination for VAR models *by Michael K. Andersson and Sune Karlsson*	2007:216
Do Central Banks React to House Prices? *by Daria Finocchiaro and Virginia Queijo von Heideken*	2007:217
The Riksbank's Forecasting Performance *by Michael K. Andersson, Gustav Karlsson and Josef Svensson*	2007:218
Macroeconomic Impact on Expected Default Freqency *by Per Åsberg and Hovick Shahnazarian*	2008:219
Monetary Policy Regimes and the Volatility of Long-Term Interest Rates *by Virginia Queijo von Heideken*	2008:220
Governing the Governors: A Clinical Study of Central Banks *by Lars Frisell, Kasper Roszbach and Giancarlo Spagnolo*	2008:221
The Monetary Policy Decision-Making Process and the Term Structure of Interest Rates *by Hans Dillén*	2008:222
How Important are Financial Frictions in the U S and the Euro Area *by Virginia Queijo von Heideken*	2008:223
Block Kalman filtering for large-scale DSGE models *by Ingvar Strid and Karl Walentin*	2008:224
Optimal Monetary Policy in an Operational Medium-Sized DSGE Model *by Malin Adolfson, Stefan Laséen, Jesper Lindé and Lars E. O. Svensson*	2008:225
Firm Default and Aggregate Fluctuations *by Tor Jacobson, Rikard Kindell, Jesper Lindé and Kasper Roszbach*	2008:226

Re-Evaluating Swedish Membership in EMU: Evidence from an Estimated Model *by Ulf Söderström*	2008:227
The Effect of Cash Flow on Investment: An Empirical Test of the Balance Sheet Channel *by Ola Melander*	2009:228
Expectation Driven Business Cycles with Limited Enforcement *by Karl Walentin*	2009:229
Effects of Organizational Change on Firm Productivity *by Christina Håkanson*	2009:230
Evaluating Microfoundations for Aggregate Price Rigidities: Evidence from Matched Firm-Level Data on Product Prices and Unit Labor Cost *by Mikael Carlsson and Oskar Nordström Skans*	2009:231
Monetary Policy Trade-Offs in an Estimated Open-Economy DSGE Model *by Malin Adolfson, Stefan Laséen, Jesper Lindé and Lars E. O. Svensson*	2009:232
Flexible Modeling of Conditional Distributions Using Smooth Mixtures of Asymmetric Student T Densities *by Feng Li, Mattias Villani and Robert Kohn*	2009:233
Forecasting Macroeconomic Time Series with Locally Adaptive Signal Extraction *by Paolo Giordani and Mattias Villani*	2009:234
Evaluating Monetary Policy *by Lars E. O. Svensson*	2009:235
Risk Premiums and Macroeconomic Dynamics in a Heterogeneous Agent Model *by Ferre De Graeve, Maarten Dossche, Marina Emiris, Henri Sneessens and Raf Wouters*	2010:236
Picking the Brains of MPC Members *by Mikael Apel, Carl Andreas Claussen and Petra Lennartsdotter*	2010:237
Involuntary Unemployment and the Business Cycle *by Lawrence J. Christiano, Mathias Trabandt and Karl Walentin*	2010:238
Housing collateral and the monetary transmission mechanism *by Karl Walentin and Peter Sellin*	2010:239
The Discursive Dilemma in Monetary Policy *by Carl Andreas Claussen and Øistein Røisland*	2010:240
Monetary Regime Change and Business Cycles *by Vasco Cúrdia and Daria Finocchiaro*	2010:241
Bayesian Inference in Structural Second-Price common Value Auctions *by Bertil Wegmann and Mattias Villani*	2010:242
Equilibrium asset prices and the wealth distribution with inattentive consumers *by Daria Finocchiaro*	2010:243
Identifying VARs through Heterogeneity: An Application to Bank Runs *by Ferre De Graeve and Alexei Karas*	2010:244
Modeling Conditional Densities Using Finite Smooth Mixtures *by Feng Li, Mattias Villani and Robert Kohn*	2010:245
The Output Gap, the Labor Wedge, and the Dynamic Behavior of Hours *by Luca Sala, Ulf Söderström and Antonella Trigari*	2010:246
Density-Conditional Forecasts in Dynamic Multivariate Models *by Michael K. Andersson, Stefan Palmqvist and Daniel F. Waggoner*	2010:247
Anticipated Alternative Policy-Rate Paths in Policy Simulations *by Stefan Laséen and Lars E. O. Svensson*	2010:248
MOSES: Model of Swedish Economic Studies *by Gunnar Bårdsen, Ard den Reijer, Patrik Jonasson and Ragnar Nymoen*	2011:249
The Effects of Endogenuos Firm Exit on Business Cycle Dynamics and Optimal Fiscal Policy *by Lauri Vilmi*	2011:250
Parameter Identification in a Estimated New Keynesian Open Economy Model *by Malin Adolfson and Jesper Lindé*	2011:251
Up for count? Central bank words and financial stress *by Marianna Blix Grimaldi*	2011:252

Wage Adjustment and Productivity Shocks *by Mikael Carlsson, Julián Messina and Oskar Nordström Skans*	2011:253
Stylized (Arte) Facts on Sectoral Inflation *by Ferre De Graeve and Karl Walentin*	2011:254
Hedging Labor Income Risk *by Sebastien Betermier, Thomas Jansson, Christine A. Parlour and Johan Walden*	2011:255
Taking the Twists into Account: Predicting Firm Bankruptcy Risk with Splines of Financial Ratios *by Paolo Giordani, Tor Jacobson, Erik von Schedvin and Mattias Villani*	2011:256
Collateralization, Bank Loan Rates and Monitoring: Evidence from a Natural Experiment *by Geraldo Cerqueiro, Steven Ongena and Kasper Roszbach*	2012:257
On the Non-Exclusivity of Loan Contracts: An Empirical Investigation *by Hans Degryse, Vasso Ioannidou and Erik von Schedvin*	2012:258
Labor-Market Frictions and Optimal Inflation *by Mikael Carlsson and Andreas Westermark*	2012:259
Output Gaps and Robust Monetary Policy Rules *by Roberto M. Billi*	2012:260
The Information Content of Central Bank Minutes *by Mikael Apel and Marianna Blix Grimaldi*	2012:261
The Cost of Consumer Payments in Sweden *by Björn Segendorf and Thomas Jansson*	2012:262
Trade Credit and the Propagation of Corporate Failure: An Empirical Analysis *by Tor Jacobson and Erik von Schedvin*	2012:263
Structural and Cyclical Forces in the Labor Market During the Great Recession: Cross-Country Evidence *by Luca Sala, Ulf Söderström and AntonellaTrigari*	2012:264
Pension Wealth and Household Savings in Europe: Evidence from SHARELIFE *by Rob Alessie, Viola Angelini and Peter van Santen*	2013:265
Long-Term Relationship Bargaining *by Andreas Westermark*	2013:266
Using Financial Markets To Estimate the Macro Effects of Monetary Policy: An Impact-Identified FAVAR* *by Stefan Pitschner*	2013:267
DYNAMIC MIXTURE-OF-EXPERTS MODELS FOR LONGITUDINAL AND DISCRETE-TIME SURVIVAL DATA *by Matias Quiroz and Mattias Villani*	2013:268
Conditional euro area sovereign default risk *by André Lucas, Bernd Schwaab and Xin Zhang*	2013:269
Nominal GDP Targeting and the Zero Lower Bound: Should We Abandon Inflation Targeting?* *by Roberto M. Billi*	2013:270
Un-truncating VARs* *by Ferre De Graeve and Andreas Westermark*	2013:271
Housing Choices and Labor Income Risk *by Thomas Jansson*	2013:272
Identifying Fiscal Inflation* *by Ferre De Graeve and Virginia Queijo von Heideken*	2013:273
On the Redistributive Effects of Inflation: an International Perspective* *by Paola Boel*	2013:274
Business Cycle Implications of Mortgage Spreads* *by Karl Walentin*	2013:275

Sveriges Riksbank
Visiting address: Brunkebergs torg 11
Mail address: se-103 37 Stockholm

Website: www.riksbank.se
Telephone: +46 8 787 00 00, Fax: +46 8 21 05 31
E-mail: registratorn@riksbank.se