Simulated Annealing Multiplicative Weights Algorithm for solving a DSGE Model

Seokil Kang*

April 2, 2019

Click for the lastest version

Abstract

This paper introduces a simulation-based adaptive algorithm to solve a DSGE model with a large state space, namely the curse of dimensionality. It aims to generate a stationary distribution over policy space which is concentrated on the optimal policy. The key strategy is to construct a finite policy space of heuristic policies. To update the distribution over policy space, the method adopts on-line computation via iterative simulation with emphasis on rolling-horizon control to foster the speed of algorithm. Subsequently, I deliver that the algorithm achieves theoretical convergence to the optimal value function and the stationary distribution over policy space is concentrated on the optimal policy. Application to solve the simple two-period RBC model follows as a sample exercise. The result shows the performance is desirable within the feasible number of iterations and size of restricted policy space respectively.

*Department of Economics, Indiana University Bloomington, sk86@iu.edu; I am indebted to my advisor, Todd Walker for much advice and guidance at all stages of this project. I also thank Bulent Guler and seminar participants at the Jordan River Economics Conference Indiana University. All remaining errors are my own.

1 Introduction

This paper introduces a simulation-based solution method for a *Dynamic Stochastic General Equilibrium* (DSGE) model to overcome *the curse of dimensionality*. The curse of dimensionality in macroeconomics refers a case in which a large state space precludes a macroeconomics model from computing its solution and estimation. Because the computational load grows exponentially for each additional state variable. This implies that solution method like value function iteration which constructs state space grids by tensor product cannot escape from the curse.

To deal with this issue, the new method switches the order of numerical solution steps, which is called as *on-line computation* in computational science literature(Chang et al., 2013). For example, value function and policy function iteration methods pre-compute policies for all states and then select one of the stored policies by observing a realized state. On the other hand, on-line computation first observes a state realized and then computes the best action(policy) to be taken. It unloads the computational burdens significantly because it only accounts for the single observed state on-line at the decision time.

The new algorithm of this paper differs not only from the mainstream of numerical solution techniques like perturbation and projection method, but also from the existing stochastic simulation method in the field. To begin with, perturbation method approximates the solution with the Taylor expansion. It provides tractable local solutions of the model with high dimension. Hence it is not a valid option for models with occasionally binding constraints. Projection method shares a very similar spirit with OLS regression. It projects the global solution on a set of function basis to minimize the residual of first order conditions. It has many advantages yet it needs to discretize the state space, meaning that the method is exposed to the large state space issue. Consequently the two popular methods are not suitable to the model which carries a large state space and occasionally binding constraints at the same time. In this context, the class of stochastic simulation methods have arisen to address the state space of a model effectively. The existing stochastic simulation methods focus on reducing the size of a state space[1]. On the other hand, I suggest a simulation method which searches for the optimal policy via on-line computation with restricted policy space. It is labeled as *Simulated Annealing Multiplicative Weighting* (SAMW) algorithm by Chang et al. (2013).

[1] Judd et al. (2011) and Maliar and Maliar (2015) review the class of the stochastic simulation methods.

The key concept of the SAMW algorithm is the policy space. The main task of SAMW is to sequentially evaluate the performance of each policy elements in the space. However it does not indicate that the evaluation should be done for all states. The idea of simulation is to compute the optimal value in terms of sample analog of the true structure. Because a sufficient number of random simulations will effectively cover the state space in Monte Carlo fashion. Therefore if one could afford a pertinent policy space, the size of a state space would be no longer a problem in running-time complexity perspective.

The SAMW algorithm has a clear structure for improvement in terms of accuracy and robustness compared to other stochastic simulation methods. Because it is associated with theories which provide convergence of the performance along with its formal evaluation. One of the theorems characterizes the error bound between the approximate of the value function and the optimal value function by the length of the horizon of the model and the discount rate. The other theorem states that the sequence of policy distribution generated by SAMW algorithm converges to a stationary distribution. Subsequently, it concludes that SAMW algorithm is *asymptotically efficient(fastest)*; The algorithm provides a *finite-time* upper bound for the sample mean of an optimal policy which converges to the optimal value function(Chang et al., 2013).

The final objective of this project is to implement the new algorithm to solve and estimate the macroeconomics model at the frontier, the model with a large state space and or occasionally binding constraints. As a first step, this paper explores the SAMW algorithm through the simple two-period Real Business Cycle (RBC) model. Through the exercise, I report the performance of the method compared to analytic solution. Though parsimonious, the practice marks some notable results of the new methodology. First, the policy distribution successfully converges to a stationary distribution concentrated on the optimal policy as the theory predicted. Secondly the performance is robust to both the construction of policy space and iteration number of simulation.

The paper is organized as follows: In section 2, a generic RBC model is elaborated as a case of a MDP. The main algorithm is presented in section 3 and the application of it follows in section 4. Section 5 concludes. Technical details are provided in Appendix. I indicate that the Section 2 and 3 refer to the basis of Chang et al. (2013).

2 Generalizing RBC Model to Markov Decision Process

To apply the SAMW algorithm as a solution method, we need to generalize a RBC model to a form of *Markov Decision Process*(MDP). A MDP is defined by five-tuple $(X, \mathcal{A}, \mathcal{A}(\cdot), P, R)$. For a detailed description, let's match each component of MDP to that of a simple RBC model with $T > 1$ periods as follows,

$$\max \mathbb{E}_0 \sum_{t=0}^{T-1} \beta^t u(c_t)$$

$$\text{s.t.} \quad c_t + k_{t+1} = A_t f(k_t) + (1-\delta)k_t \tag{1}$$

$$A_t \sim \text{ a Markov process} \tag{2}$$

$$k_0 \text{ is given} \tag{3}$$

where $u(\cdot)$ is an utility function and $f(\cdot)$ is a production function. c_t, k_t and A_t are consumption, capital stocks and total factor productivity at period $t = 0, \ldots, T-1$ respectively. The parameter $\beta \in (0,1)$ is a time discount rate, $\alpha \in (0,1)$ is the usual parameter of capital in Cobb-Douglas production function, and $\delta \in [0,1]$ is a depreciation rate of capital.

The first component X is a *state space* of states $x \in X$. For the example model, a state is denoted as $x = (k, A)$ or $x_t = (k_t, A_t)$ for all time period $t = 0, \cdots, T-1$. Secondly, $\mathcal{A}$ is an *action space*. Its element $a \in \mathcal{A}$ is an action or a control variable in terms of dynamic programming. The agent of this simple RBC model decides her consumption and capital stocks to hold for each period, *i.e.* $a_t = (c_t, k_{t+1})$. We can replace one between c_t and k_{t+1} with resource constraint (1). In fact, setting $a_t = k_{t+1}$ as an action for period t is convenient to describe an admissible action space. An *admissible action space* $\mathcal{A}(x)$ is a subset of action space which collects admissible actions at the given state $x \in X$. By admissibility, it means that an action satisfies constraints of a model. Therefore the admissible action space for the example model becomes $\mathcal{A}(x_t) = [0,\ A_t f(k_t) + (1-\delta)k_t]$ for period t[2]. Next, P is a *probability of state transition*. $P(x,a)(x')$ denotes the probability of transitioning from a state $x \in X$ to another state $x' \in X$ when taking an action $a \in \mathcal{A}$. A Markov matrix is an equivalent transition function familiar to the macroeconomics literature. Lastly, R is a *reward function* of a state and action. An utility function $u(x,a)$ directly maps an instant reward from the current state x and the given

[2]Thus at period t, the capital stock of current period k_t is a state and the next period stock k_{t+1} is an action.

action a. An utility function $u(c)$ can be rewritten with the resource constraint (1) as $u(x,a) = u((k,A),k') = u(Af(k)+(1-\delta)k-k')$. Therefore, replacing c_t with the budget constraint (1) gives

$$\max \mathbb{E}_0 \sum_{t=0}^{T-1} \beta^t u\big(A_t f(k_t) + (1-\delta)k_t - k_{t+1}\big)$$

$$\text{s.t.} \quad A_t \sim \text{ some stochastic(Markov) process} \tag{4}$$

$$k_0 \text{ is given} \tag{5}$$

Solving a DSGE model is to answer the following question; what should we do in a given situation? In other words, it is to find a certain *policy* $\pi = \{\pi_t\}_{t=0}^{T-1}$, a time-sequence of actions to take for each given state. Such policy at t is a function of states which maps all states to the admissible action, *i.e.* $\pi_t : X \to \mathcal{A}(\cdot)$. The optimal policy π^* is defined as the policy maximizing present value of expected utility subject to given constraints of a model. Applying the concept of a policy allows us to represent a MDP with respect to state x. Dynamic programming develops this idea and describes a MDP as a system of state, widely known as value function with a given policy π,

$$V^\pi(x) = \mathbb{E}\Big[\sum_{t=0}^{T-1} \beta^t R(x_t, \pi_t(x_t)) \Big| x_0 = x\Big]$$

with a given initial state x_0 and a discount rate $\beta \in (0,1)$.

A DSGE model can always be redefined as a MDP. The state space, action space, transition function and reward function are well defined in a DSGE model already. The admissible action space here is the action space incorporated with constraints of a DSGE model. Define a policy function as $k_{t+1} = \pi_t(k_t, A_t)$, then we can fit the sample RBC model into the value function for policy π as above,

$$V^\pi(k,A) = \mathbb{E}\Big[\sum_{t=0}^{T-1} \beta^t u\big(A_t f(k_t) + (1-\delta)k_t - \pi_t(k_t,A_t)\big) \Big| (k_0,A_0) = (k,A)\Big] \tag{6}$$

To simplify the notation, let's denote state $x_t = (k_t, A_t)$, action $a_t = k_{t+1}$, the reward function combined with constraint $u(x_t,a_t) = u\big(A_t f(k_t) + (1-\delta)k_t - a_t\big)$ and $u(x_t, \pi_t(x_t)) = u\big(A_t f(k_t) +$

$(1-\delta)k_t - \pi_t(k_t, A_t)\big)$ from now on. Then we can rewrite the value function of policy π,

$$V^{\pi}(x) = \mathbb{E}\Big[\sum_{t=0}^{T-1} \beta^t u(x_t, \pi_t(x_t))\Big| x_0 = x\Big] \tag{7}$$

After transforming a DSGE model into the value function representation, we can define the optimal value function,

$$V^*(x) = \sup_{\pi \in \Pi} V^{\pi}(x) \tag{8}$$

where Π is a set of policies $\pi = \{\pi_t\}_{t=0}^{T-1}$. It is familiar to write a value function for a policy π for all $x \in X$ recursively as

$$V^{\pi}(x) = u(x, \pi(x)) + \beta \sum_{x' \in X} P(x, \pi(x))(x') V^{\pi}(x') \tag{9}$$

$$V^{\pi}(x) = u(x, \pi(x)) + \beta \mathbb{E}[V^{\pi}(x')|x] \tag{10}$$

and the optimal value function for all $x \in X$ as

$$V^*(x) = \sup_{a \in \mathcal{A}(x)} \big\{u(x,a) + \beta \sum_{x' \in X} P(x,a)(x') V^*(x')\big\} \tag{11}$$

$$V^*(x) = \sup_{a \in \mathcal{A}(x)} \big\{u(x,a) + \beta \mathbb{E}[V^*(x')|x]\big\} \tag{12}$$

Note that the value function is a function of state x, not of policy π. It implies that a value function merely gives a measure to evaluate the given policy π but a direction of improvement of policy. To solve for π^*, we need to derive the optimality conditions from the first order conditions and envelop theorem. Unfortunately such analytic approach is hardly available in most DSGE models due to their complexity. In this regard, several numerical methods have been developed to exploit the properties of a value function and its associated optimal policy.

3 Simulated Annealing Multiplicative Weights Algorithm

To undermine the curse of dimensionality the class of stochastic simulation methods has been established to address the state space of a model effectively. Generally they have focused on deweighting the state space by computing only in the part of the state space which is visited in

equilibrium(Chang et al., 2013, Judd et al., 2011). On the other hand, I suggest a simulation method called SAMW which searches for the optimal policy via on-line computation with restricted policy space. This section is followed by the mechanism description of the algorithm. First, section 3.1 categorizes a simulation method as an *one-line computation*, a concept from the control theory literature. It supports the idea that the simulation method can treat a large state space in feasible manner. Section 3.2 introduces *rolling-horizon control* also known as *model predictive control*. It is the main strategy of the algorithm to track the optimal policy for a MDP with infinite horizon. Finally 3.3 walks through the SAMW algorithm in details.

3.1 On-line Simulation-Based Control

The usual solution method to compute the optimal value function V^* is value function iteration. It starts with an initial guess $V^{(0)}$ to feed in and then searches for an optimal $a \in \mathcal{A}(x)$. It iterates 'feed-in and search' until $V^{(i)}$ converges to $V^{(i+1)}$ for all state $x \in X$. Naturally it is vulnerable to the size of the state space X. Policy function iteration consists of two subroutines: policy evaluation and policy improvement. For any policy $\pi \in \Pi$, one can compute a value function by solving (9) for fixed point function V^π over all $x \in X$. That is, there are $|X|$ number of linear equations for the same number of unknowns, $V^\pi(x)$. Next, policy improvement attains a new policy $\hat{\pi}$ by

$$\hat{\pi}(x) \in \arg\sup_{a \in \mathcal{A}(x)} \{u(x,a) + \beta \sum_{x' \in X} P(x,a)(x') V^\pi(x')\} \tag{13}$$

and these two steps iterate until V^π converges to $V^{\hat{\pi}}$.

Both iterative methods impose computation over the whole state space X. Inevitably their running-time complexity heavily depends on the state space size for both cases[3]. In the control theory literature, there are two methods of computation, off-line and on-line computation(Chang et al., 2013). Policy iteration and value function iteration are the examples of off-line computation. They compute each policies for all state points in the state grid and then select an optimal policy after the realization of a certain state. This is computationally heavy as the number of state increases in exponential manner-the curse of dimensionality.

On-line computation reverses this step. It first let the system of interest reaches a certain state and then compute the optimal policy. A simulation method conducts on-line computation

[3] Chang et al. (2013) demonstrates the running-time complexity of these iteration methods. Each single iteration's time complexity is $\mathcal{O}(|X|^2|\mathcal{A}|)$ for value function iteration and $\mathcal{O}(|X|^2|\mathcal{A}| + |X|^3)$ for policy function iteration. Note that the curse of dimension arises in the tensor product of a state space X.

by generating the stochastic process of a MDP. It can bring up a sample path of state of a model through a number of simulation seeds $\{w_t^i\}_{t=0}^{T-1}$ for $i = 1, \ldots, N$. Let's take the simple T period RBC model as an illustration example. One can easily generate the series of total factor productivity process $\{A_t\}_{t=0}^{T-1}$ based on the given model (2). This suffices for computing a sample path of value function for policy π. That is, we only need to compute for the realized state, not for the whole state space. This idea turns our task to find a computation method of which finite sample paths sufficiently covers the given state space instead of enumerating a state space. Suppose that there exists a finitely large iteration number of simulation N such that sufficiently covers all possible histories of stochastic evolution of the system. Then the sample paths produced by such simulation will be a proper sample analog to estimate the value function (7),

$$\hat{V}^{\pi}(x) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\beta^t u(x,\pi(x))\big|x_0 = x, \{w_t\}_{t=0}^{T-1}\right)$$

$$\hat{V}^{\pi}(x) \underset{N\to\infty}{\to} V^{\pi}(x)$$

The beauty of a simulation method is embedded in switching the order of maximization and expectation operator of a dynamic programming. For the original problem (10), it requires to derive the maximum of the expectation operator term conditional on state $x \in X$. Thus we have to compute $\max\{\mathbb{E}[V(x')|x]\}$ either analytically or numerically for all states in X. Meanwhile a simulation method seeks a way to approximate such objective by switching the order of operators to $\mathbb{E}[\max\{u(x,a)+V(x')\}|x]$. It is a more desirable case because it does not employ any maximization routine, a main culprit of both running-time complexity and instability in most cases. In particular, the SAMW algorithm omits a maximization routine but only generates a sequence of weight distribution over the policy space. It reaches the optimal policy and value function by simply updating the weight of each policy over the policy space through iterative simulations.

3.2 Rolling-Horizon Control

A simulation method is not directly applicable for solving a DSGE model with infinite-horizon since there is no final period to terminate the process. As a result, approximating an infinite-horizon model to a finite-horizon version is necessary to solve it through a simulation method. It is true that the longer the horizon of the model is, the better it approximates. However it is not necessarily a good idea to make the horizon of the model

as long as possible. Because this comes with a cost of computational load by construction. Hence we need to settle the trade-off between accurate approximation and computational cost formally.

Such approximation framework is known as rolling-horizon control from control theory literature(Chang et al., 2013). It demonstrates a theoretical error bound between the stationary solution of infinite-horizon model and the optimal initial policy of truncated model, parameterized by the horizon length and discount rate[4]. Intuitively with sufficiently long horizon and small discount rate, the stationarity of infinite horizon model can be estimated relatively well by the early part of truncated model. That is, the nonstationary optimal policy of the initial period becomes a good approximate of the original solution because the error bound diminishes as the horizon gets longer given a discount rate lower than one.

This allows us to focus on the policy function of the initial period as we are going to roll over the horizon. Suppose that we were able to extend the horizon of the model to a fairly long period, say T periods. Then we get T nonstationary optimal policy functions, $\{\pi_t^*\}_{t=0}^{T-1}$. Chang et al. (2013) shows that this length T model's policy function of initial period would become the approximate of the stationary policy function of the infinite horizon. The rationale behind is that once the agent plans for T period ahead, after a period later, she can restart her plan again. In this way, only the initial period policy will be applied repeatedly and it eventually becomes the stationary solution of the model. This stationarity is very beneficial because keeping the simulation's framework constantly can save a great computational expense. The matter is that the agent should not be worse off too much from truncating her plan up to T period. Therefore we can infer that the approximation would depend on T and the discount rate β.

To denote of the dependency of value function on finite horizon T, let's rewrite (7) as

$$V_T^{\pi}(x) = \mathbb{E}\left[\sum_{t=0}^{T-1} \beta^t u(x_t, \pi_t(x_t)) \middle| x_0 = x\right] \tag{14}$$

and the optimal value function as

$$V_T^*(x) = \sup_{\pi \in \Pi} V_T^{\pi}(x) \tag{15}$$

[4]This is not an alien concept for the dynamic macroeconomics because the transversality condition of infinite-horizon model has been taking this role already.

In addition, let $B(X)$ be the space of bounded real-valued functions on X. For $\Phi \in B(X)$, $x \in X$, define an operator $T : B(X) \to B(X)$ by

$$T(\Phi)(x) = \sup_{a \in \mathcal{A}(x)} \{u(x,a) + \mathbb{E}[\Phi(x')|x]\} \tag{16}$$

Then we can define a *rolling-T-horizon control policy* $\pi_{rh} \in \Pi$ as a stationary policy for the infinite horizon problem from an optimal non-stationary policy $\{\pi_t^*\}_{t=0}^{T-1}$ for the finite-horizon problem (15) of length $T < \infty$, by taking $\pi_{rh} = \pi_0^*$, for a given initial state $x_0 = x$. *i.e.* it satisfies

$$T_{\pi_{rh}}(V_T^*)(x) = T(V_T^*)(x),\ x \in X \tag{17}$$

Again the logic of rolling horizon is that the decision maker can approximate her optimal policy for infinite horizon π^* with the one for the finite horizon of a sufficient length $T < \infty$, $\pi_0^* \in \{\pi_t^*\}_{t=0}^{T-1}$. For an arbitrary period t, an individual takes her action based on π_0^*. Then the next period $t+1$ comes, instead of choosing π_1^* as her rule of thumb for $t+1$, she can choose π_0^* by rolling her horizon of length T. Such rolling is stationary because the individual rolls the finite horizon forever thanks to the original setup of infinite horizon. Thus if the rolling horizon has a sufficient length and discount rate, losing the foresight from $T+1$ period is rather innocuous.

The rolling horizon control approximates the original infinite horizon MDP twice. First, i) it controls the horizon to finite T. Secondly, ii) it approximates V_T^* of finite horizon model by simulating V_T^π for some policies $\pi \in \Pi$. Hence, we can decompose the error between V^*(optimal value function of infinite horizon) and V_T^π(simulated value function with finite horizon T) into two parts;

Theorem 3.1. *(Chang et al., 2013, Theorem 5.1) Given $V \in B(x)$ such that $|V_T^*(x) - V(x)| \leq \varepsilon$ for all $x \in X$, consider a policy $\pi \in \Pi_s$(stationary policy set) such that $T_\pi(V) = T(V)$. Then,*

$$0 < V^*(x) - V^\pi(x) \leq \frac{\bar{u}}{1-\beta}\beta^T + \frac{2\beta\varepsilon}{1-\beta}, \quad x \in X \tag{18}$$

where $\bar{u}$ is the upper bound on instant utility[5].

[5]the maximum reward $\bar{u}$ is bounded in both finite and infinite horizon cases thanks to the transversality condition. We can always posit a finite bound $\bar{u}$ under admissible action space $\mathcal{A}(x)$ for general utility maximization case.

The first part of the error terms diminishes as the horizon length T increases with a given discount rate $\beta < 1$. The second part mainly depends on how accurately the certain policy π mimics the optimal policy π^*, denoted as ε. Hence the above theorem approves the rolling horizon control as a numerical strategy, which holds for a simulation method in general.

3.3 Main Mechanism Description The heart of this adaptive algorithm is at devising a finite, heuristic subset of policy space $\Lambda \subset \Pi$. To understand why, suppose that we have designed a finite-subset Λ with some finite, heuristic policies. Then our objective is to compute the optimal value function

$$V^* = \max_{\pi \in \Lambda} V^\pi \tag{19}$$

The SAMW algorithm carries out its task by generating a sequence of distributions over Λ. First it starts with equal probability weighting on every policy, a uniform distribution. Then for each iteration it generates a random sequence to simulate the model and compute the value function for each policy. Because the optimal value function is the one with maximum value, the SAMW algorithm updates the policy distribution with V^π for all $\pi \in \Lambda$ by a simple multiplicative weighting rule. This 'compute & update' iteration makes the policy distribution concentrated on the optimal policies. In the end, the sequence of distribution converges to a stationary distribution with a proper annealing parameter; It is the completion of the SAMW algorithm.

Note that a single iteration only requires $|\Lambda|$ sample paths. So it is independent to the size of the state space. Each evaluation need not to be done for all states. On-line computation via simulation is to compute the optimal value function in terms of sample analog of the true structure. Thus if we could malleate a pertinent policy space, the size of a state space would be no longer a problem from a running-time complexity perspective. In fact, the running-time complexity of each iteration of SAMW algorithm is $\mathcal{O}(|\Lambda|T)$ because it evaluates all polices in Λ of length T. Of course it requires more iterations as the number of state increases, however it does not fall into the curse of dimensionality. Because running-time complexity of the SAMW algorithm does not grow exponentially to the state space size. The main source of the curse is the tensor product in constructing a state space.

3.3.1 Sequence of Policy Distribution

Recall that the optimal policy in a stochastic economy is a distribution, not a point. In this sense, what we are looking for is a certain moment(mean or median) of the distribution, just like a median of impluse response function in Vector Autoregression analysis. Define Φ as the set of all probability distributions over Λ and denote $\phi(\pi)$ as the probability for policy π. Through the algorithm, the sequence of distributions will be concentrated on the optimal policies. Denote i for each simulation iteration, then a sample path estimate of the value function with a policy π becomes

$$V_i^{\pi}(x) = \sum_{t=0}^{T-1} \beta^t u(x_t, \pi_t(x_t)) | x_0 = x, \{w_t^i\}_{t=0}^{T-1}) \tag{20}$$

The probability distribution over Λ will be updated at each iteration i. The iterative updating rule for the distribution ϕ^{i+1} from ϕ^i and its associated sample path $\{V_i^{\pi}\}$ follows

$$\phi^{i+1}(\pi) = \phi^i(\pi) \frac{\gamma_i^{V_i^{\pi}}}{Z^i} \quad \forall \pi \in \Lambda \tag{21}$$

where $\gamma_i > 1$, $\forall i = 1, \ldots, N$ is a tuning parameter of the algorithm, the normalization factor Z^i is given by $Z^i = \sum_{\pi \in \Lambda} \phi^i(\pi) \gamma_i^{V_i^{\pi}}$ and the initial distribution ϕ^1 is an uniform distribution for each $\pi \in \Lambda$. The parameter γ_i takes the same roll as the tempering parameter in the generic *Sequential Monte Carlo*(SMC) algorithm. The algorithm sets $\{\gamma_i\}$ as a monotonically decreasing sequence converging to one. This allows the updating scale $\gamma_i^{V_i^{\pi}}$ to vary freely at the relatively small i. However it cools down the abrupt revision of the distribution as i increases by construction. It helps the sequence of the distribution $\{\phi^i\}$ converge to a stationary distribution. This is why the tuning parameter γ_i is called as *annealing* parameter.

3.3.2 Convergence Analysis

The following lemmas build up the convergence theorem for the SAMW algorithm. The first lemma below states that the SAMW algorithm with the updating rule (21) provides the finite-time upper bound for the sample mean of V^*.

Lemma 3.2. *(Chang et al., 2013, Lemma 5.2) For $\phi \in \Phi$, define*

$$\bar{V}_i(\phi^i) = \sum_{\pi \in \Lambda} V_i^{\pi} \phi(\pi) \tag{22}$$

$$\Psi_N^{\pi^*} = \frac{1}{N} \sum_{i=1}^{N} V_i^{\pi} \tag{23}$$

For $\gamma_i = \gamma > 1$, $i = 1, \ldots, N$, the sequence of distribution $\{\pi^i\}_{i=1}^N$ generated by SAMW with (21) satisfies (a.s.)

$$\Psi_N^{\pi^*} \leq \frac{\gamma - 1}{\ln \gamma} \frac{1}{N} \sum_{i=1}^{N} \bar{V}_i(\phi^i) + \frac{\ln |\Lambda|}{N \ln \gamma} \tag{24}$$

for any optimal policy $\pi^* \in \Lambda$

The lemma has clear interpretation; The sample mean of value function for a certain policy (23) has an upper bound. Such upper bound is *finite-time* in the sense that the upper bound terms are decreasing *a.s.* in the iteration number N. One can easily see that the two parts of the upper bound (24) have the limit as $\lim_{\gamma \to 1+} \frac{\gamma - 1}{\ln \gamma} = 1$ and $\lim_{N \to \infty} \frac{\ln |\Lambda|}{N \ln \gamma} = 0$ for a finite $|\Lambda|$ and $\gamma \to 1+$. This implies that if we can afford a more expensive simulation setup(larger N), the upper bound gets closer to the sample mean of $\bar{V}$. (24) also suggests that the convergence depends on the size of restricted policy space and annealing parameter γ as well. In fact, setting a proper annealing parameter matters for the sequence $\{\phi^i\}$ to converge, which is the statement of the following lemma.

Lemma 3.3. *(Chang et al., 2013, Lemma 5.3) Define the total variation distance for probability mass functions p and q by*

$$d_N(p, q) = \sum_{\pi \in \Lambda} |p(\pi) - q(\pi)| \tag{25}$$

Let $\{\psi(N)\}$ be a decreasing sequence such that $\psi(N) > 1 \; \forall N$ and $\lim_{N \to \infty} \psi(N) = 1$. For $\gamma_i = \psi(N)$, $i = 1, \ldots, N + k$, $k \geq 1$, the sequence of distribution $\{\phi^i\}$ generated by SAMW with (21) satisfies (a.s.)

$$\lim_{N \to \infty} d_N(\phi^N, \phi^{N+k}) = 0 \tag{26}$$

The lemma above regulates a proper annealing parameter as a sequence that monotonically decreases and converges to one. One example which satisfies the given conditions is $\gamma_i(N) = 1 + \sqrt{\frac{1}{N}} \; N > 0$. A constant parameter $\gamma_i = \gamma > 1 \; \forall i = 1, \ldots, N$ also feasible as well[6].

Finally, the previous lemmas constitute the convergence theorem which states that the upper bound (24) converges to the optimal value function with a proper annealing parameter.

[6] I verified the convergence in the practice exercise with $\gamma_i = 2 \; \forall i = 1, \ldots, N$.

Theorem 3.4. *(Chang et al., 2013, Theorem 5.4) Let* $\{\psi(N)\}$ *be a decreasing sequence such that* $\psi(N) > 1 \, \forall N$ *and* $\lim_{N\to\infty} \psi(N) = 1$ *and* $\lim_{N\to\infty} N\psi(N) = \infty$. *For* $\gamma_i = \psi(N)$, $i = 1, \ldots, N+k$, $k \geq 1$, *the sequence of distribution* $\{\phi^i\}$ *generated by SAMW with* (21) *satisfies (a.s.)*

$$\frac{\gamma - 1}{\ln \gamma} \frac{1}{N} \sum_{i=1}^{N} \bar{V}_i(\phi^i) + \frac{\ln |\Lambda|}{N \ln \gamma} \to V^* \tag{27}$$

and $\phi^i \to \phi^* \in \Phi$, *where* $\phi^*(\pi) = 0$ *for all* π *such that* $V^\pi < V^*$.

3.3.3 Approach to Restricting Policy Space One can plainly see that constructing the restricted policy space Λ is the key to the success of the SAMW algorithm. To begin with, Λ must contain the optimal policy π^* and minimize the size of the space. There are few implementable ideas which are not exhaustive. They meet the requisitions above to some degree and can be combined together immediately. Discretizing the admissible action space $\mathcal{A}(x)$ is the first thing that comes to mind. It is generally applicable analogously to the case of discretizing the state space. It is a safe approach in the sense that even if the actual optimal policy fails to be included as one of the grids, its neighborhood certainly be a good approximate. However the finer the policy grid is, the heavier the computation load becomes.

Adapting the resampling subroutine of SMC type algorithm is another treatment to lighten Λ. Herbst and Schorfheide (2015) introduces generic SMC algorithm of three main subroutines. A selection step is one of them and it resamples the particles proportional to their weights. In this way, particles with very low likelihood cannot be drawn so that the distribution of particles converges to the stationary one effectively. Likewise, the distribution $\{\phi^i\}$ over the restricted policy space can take the same role here. With certain iteration periods, we can drop out some policies in Λ by setting up a threshold weight to prune Λ.

Employing the notion of economics theories is always a compelling way to address the restricted policy space Λ. Notice that the SAMW algorithm is designed for a generic MDP, not just for a DSGE model of macroeconomics. That is, the method has not yet exploited the inherent features of economics, the equilibrium conditions. Each condition relates optimal polices to each other as a set. For instance, solving a consumption-saving problem usually boils down to find the solution of initial period consumption and the intertemporal Euler equation enumerates the optimal policy of consumption. This tells that if one posits consumption policy of initial period, the subsequent policies are one-to-one related through the intertemporal Euler

equation. It dramatically reduces the policy space compared to the case of discretizing action space of each period.

Finally a technical remark is that the SAMW algorithm is *asymptotically efficient(fastest)*. It means that the algorithm provides a *finite-time* upper bound (24) for the sample mean of an optimal policy $\pi^* \in \Lambda$ and the upper bound converges to the optimal value function V^* with $\mathcal{O}(\frac{1}{\sqrt{N}})$ of running-time complexity(Chang et al., 2013). So increasing N may hurt the speed of algorithm for more iteration, but the convergence speed itself gets faster at the same time.

4 Application of SAMW Algorithm

4.1 Example Consider a simple two-period centralized RBC model with a total factor productivity shock in a production function. Let's describe the model with discrete stochastic states $s \in \mathcal{S}$ in order to emphasize the state-dependency of the model and its solutions.

$$\begin{aligned} \max_{c_0, c_1(s), k_1} \; & u(c_0) + \beta \sum_{s \in \mathcal{S}} p(s) u(c_1(s)) \\ \text{s.t. } & c_0 + k_1 = A_0 f(k_0) \\ & c_1(s) = A_1(s) f(k_1) \quad \forall s \in \mathcal{S} \\ & k_0, A_0 \text{ is given} \end{aligned}$$

I assumed a complete depreciation ($\delta = 1$) to derive a closed-form solution by hand. Then the performance of the methodology studied here can be compared with the analytic solution later. We can solve the model by Lagrangian method,

$$\mathcal{L} = u(c_0) + \beta \sum_{s \in \mathcal{S}} p(s) u(c_1(s)) + \lambda_0 \{A_0 f(k_0) - c_0 - k_1\} + \sum_{s \in \mathcal{S}} \lambda_1(s) \{A_1(s) f(k_1) - c_1(s)\}$$

deriving the first order conditions with respect to c_0, $c_1(s)$ and k_1,

$$\begin{aligned} c_1 : \quad & u_c(c_0) = \lambda_0 \\ c_1(s) : \quad & \beta p(s) u_c(c_1(s)) = \lambda_1(s) \quad \forall\, s \in \mathcal{S} \\ k_1 : \quad & u_c(c_0) = \beta \sum_{s \in \mathcal{S}} p(s) u_c(c_1(s)) A_1(s) f'(k_1) \end{aligned}$$

The intertemporal Euler equation is derived by combining all three conditions above

$$u_c(c_1) = \beta \sum_{s \in \mathcal{S}} p(s) u_c(c_1(s)) A_1(s) f'(k_1) \tag{28}$$

Let's further specify the utility and production of the model with $u(c) = \ln c$ and $f(k) = k^\alpha$ where $\alpha \in (0, 1)$ is the usual parameter of capital in Cobb-Douglas production. The solution of the model can be derived by plugging the resource constraint of the final period into the intertemporal Euler equation.

$$\begin{aligned} c_0^{-1} &= \alpha\beta \sum_{s \in \mathcal{S}} p(s) A_1(s)^{-1} k_1^{-\alpha} A_1(s) k_1^{\alpha - 1} \\ &= \alpha\beta \sum_{s \in \mathcal{S}} p(s) k_1^{-1} = \alpha\beta k_1^{-1} \end{aligned} \tag{29}$$

We can solve for c_1 and k_2 by substituting one of them with (29) in the resource constraint of the beginning period and then the solution for $c_1(s)$ easily follows

$$c_0^* = \frac{1}{1+\alpha\beta} A_0 k_0^\alpha, \quad k_1^* = \frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^\alpha, \quad c_1(s)^* = A_1(s) \left(\frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^\alpha \right)^\alpha \quad \forall s \in \mathcal{S} \tag{30}$$

The optimal policy is not stationary because the model has a finite horizon.

4.2 Result Table 1 summaries the results and performances of the SAMW algorithm with various setups applied to the previous exercise. At a glance overall performance of the method is successful in the sense that majority of simulated means of policy are fairly closed to the true optimal value(.6528). It is more formal and conventional to check the accuracy of numerical solution with *Euler equation error* proposed by Judd (1992). However it requires a model to have steady state to compute the Euler error in this way. Thus I applied slightly different version of Euler equation by Den Haan and Marcet (1994) which constructs unit-free error in terms of normalized consumption[7]. It has a clear interpretation as well; An Euler error x is computed in base 10 $\log$, so the error can be measured as 10^x units of consumption goods. The fifth column of Table 1 contains Euler errors. In average more iteration usually gives smaller error less than 1% unit of consumption[8].

[7] Please refer Appendix B for Euler error computation.

[8] One notable point is the simulation setup of $|\Lambda| = 10$ yields particularly small error only at $N = 10^3$ while the rest of iteration setups gives the largest errors. In fact the Euler error with $N = 1114$ is -5.0359 and error increases

The result in Table 1 distinctly emphasizes how important to form the restricted policy space Λ properly. First, the variance of the policy distribution is decreasing with smaller policy space size and larger iteration number respectively as predicted by the previous theories. This is also true for the finite-time upper bound[9]. Combining these empirical facts with the running time records, a smaller policy space is desirable in order to increase the simulation number. At the same time one can assert that investing computational resource too much is inefficient. Because the simulated mean of optimal policy and value function are already appealing with relatively small N.

sharply for some $N > 1114$. However the other simulation setups except $|\Lambda| = 10$ showed monotonic decreasing of Euler errors as I increased N.

[9]To obtain a valid upper bound, assumption $\bar{u} < \frac{1}{N}$ to bound $o \leq V^{\pi} \leq 1 \; \forall \pi \in \Pi$ is crucial. However, several exercises report that such restriction can be relaxed. Results of Table 1 are computed with assuming $\inf_{\pi \in \Lambda} V\pi = -10^2$ which represents sufficiently inferior value. Alternatively, we can apply the *Inada condition* to exclude the corner policies $\pi = 0$ and $\pi = A_0 k_0^{\alpha}$ whose value function yield negative infinity. This is a good example of applying an economics theory to add up a restriction on the policy space.

		Policy Estimate			Value Function Estimate		
Simulation Number N	Policy Space Size $\|\Lambda\|$	Simulated Mean	Simulated Variance	Euler Error	Simulated Value Function	Finite-time Upper Bound	Running Time(sec)
10^2	5	.709	.0197	-.9337	.494	.6193	.0427
	10	.7118	.0587	-.9122	.5208	.6488	.0536
	100	.7124	.0578	-.9078	.4378	.8845	.0511
	500	.7124	.0578	-.9078	.4389	1.0263	.0455
10^3	5	.6786	0	-1.2785	.4997	.5687	.0538
	10	.6564	.0143	-2.1449	.4918	.5577	.0451
	100	.6706	.0171	-1.4414	.4867	.6396	.1204
	500	.6706	.0171	-1.4414	.5045	.655	1.0848
2×10^4	5	.6786	0	-1.2788	.4974	.5087	1.2229
	10	.6034	.0001	-1.012	.4924	.5093	5.3647
	100	.6567	.0037	-2.1096	.4917	.5328	68.8457
	500	.6567	.0037	-2.1096	.4978	.5166	381.8495
10^5	5	.6786	0	-1.2788	.4982	.5024	76.7664
	10	.6032	0	-1.0099	.4955	.5024	167.8472
	100	.6545	.0016	-2.4629	.4999	.5116	1852.172
	500	.6545	.0016	-2.4629	.4985	.5116	9131.085

Simulated Mean is computed as the weighted sum of policy with stationary distribution, $\sum_{\pi\in\Lambda}\phi^N(\pi)\pi$ and variance is computed in the same fashion. The Euler error is marked in terms of logarithm of base 10. Simulated Value Function is derived with $\bar{V}_N(\phi)$ from (22) and Finite-time Upper Bound follows (24). Lastly, Running Time seconds are recorded based on MATLAB R 2018a. Calibration for model parameters follows; $\alpha = \frac{1}{3}$, $\beta = .95$, $k_0 = 20$, $A_0 = 1$. The true value of the optimal policy is $\pi^* = .6528$ and the value function is $V^* = .4973$ respectively.

Table 1: SAMW Results from Different Simulation Parameters

Figure 1 describes two simulation results with different simulation parameters. The left panel is the one with $N = 100$ and $|\Lambda| = 10$ and the right one is with $N = 100,000$ and $|\Lambda| = 500$. The vertical line depicts the true optimal policy. It clearly shows that the simulation successfully approximates the optimal policy given that both distributions are concentrating on the analytic solution as iteration proceeds. The restricted policy space with more policies shapes smoother kernel of distribution over Λ. Also larger iteration number N produces more concentrated kernel as well. However they might be achieved at a cost of running-time for a large-scale model.

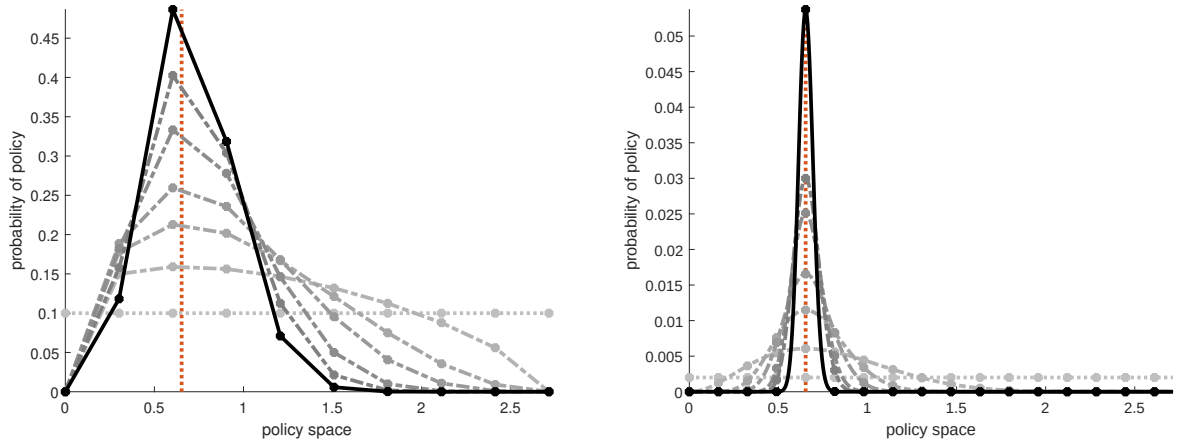

Figure 1: Policy Distributions Generated with Different Simulation Parameters($N = 100, |\Lambda| = 10$ for the left panel and $N = 100,000, |\Lambda| = 500$ for the right panel).

The sequence of distribution $\{\phi^i\}$ starts from the uniform distribution, the horizontal dotted line. Dashed lines gradually concentrating are the interim distributions, the darker the latter. The solid line is the stationary distribution ϕ^N. The red vertical dotted line points the true optimal policy.

Note that the early parts of the sequence of distribution over Λ are already clustered. We can exploit this fast concentration by pruning some policies with lower weight to reduce the size of policy space.

We can examine the performance of the algorithm by looking at the sequence of weighted sum of policy $\sum_{\pi\in\Lambda}\phi^i(\pi)\pi$ and estimated value for iteration i $\bar{V}_i(\phi^i)$. Figure 2 illustrates the simulation performance in lines with iteration number. A prospective feature from Figure 2 is that the required iteration number to simulate the optimal policy is practically small $(N < 1,000)$ and the convergence is fast and smooth. Another benefit of the algorithm is that one can arrange it to report its performance at any iteration term.

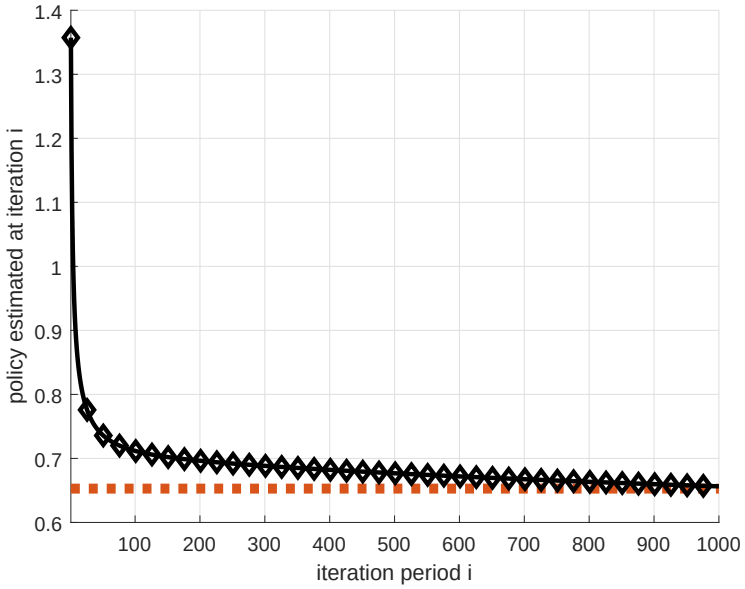

Figure 2: Performance of SAMW with $N = 1,000, |\Lambda| = 10$

The black line is the sequence of weighted sum of policy $\sum_{\pi \in \Lambda} \phi^i(\pi)\pi$ in every 20-period interval. The red dotted line marks the optimal policy.

5 Conclusion

Based on the literature of computational engineering, this paper has introduced an adaptive simulation algorithm to solve a DSGE model with a large state space, namely the curse of dimensionality. This imported method generates a stationary distribution over policy space which is concentrated on the optimal policy. Instead of minimizing the size of state space, the algorithm alters its strategy to construct a finite policy space of heuristic policies. To update the distribution over policy space, the method adopts on-line computation via iterative simulation with emphasis on rolling-horizon control to foster the speed of algorithm. As a result, I have delivered that the algorithm achieves theoretical convergence to the optimal value function and the stationary distribution over policy space is concentrated on the optimal policy. I also have applied this numerical method to solve the simple two-period RBC model as a sample exercise. The result shows that the performance of the method is desirable within feasible number of iterations and size of restricted policy space respectively.

One important scheme for the SAMW algorithm is about how to minimize the policy space which includes the optimal policy. There could be more than one strategies to take. One can reduce the policy space from a technical perspective; Discretizing the space and/or adap-

tively thinning out it through the feedback from the distribution sequence. At the same time, restricting the policy space with economics theories are strongly encouraged. It is easily implementable and the optimality conditions of a given model can greatly limit the policy space with sample paths solely dependent to the initial point. These measures can and should be taken jointly.

This paper marks a first step to solve and estimate the macroeconomics model at the frontier which are large-scaled and complicated in many quarters. Foremost, my anticipation of the method is to estimate a model with occasionally binding constraints. Yet the natural next steps are to apply the algorithm to solve a model with more states and periods, possibly with occasionally binding constraints later on. For those tasks, computational works should be done through a program faster than MATLAB. JULIA is the promising package to implement the method.

References

Chang, H. S., J. Hu, M. C. Fu, and S. I. Marcus (2013): *Simulation-based algorithms for Markov decision processes*, Springer Science & Business Media.

Den Haan, W. J. and A. Marcet (1994): "Accuracy in simulations," *The Review of Economic Studies*, 61, 3–17.

Herbst, E. P. and F. Schorfheide (2015): *Bayesian estimation of DSGE models*, Princeton University Press.

Judd, K. L. (1992): "Projection methods for solving aggregate growth models," *Journal of Economic Theory*, 58, 410–452.

Judd, K. L., L. Maliar, and S. Maliar (2011): "Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models," *Quantitative Economics*, 2, 173–210.

Maliar, L. and S. Maliar (2015): "Merging simulation and projection approaches to solve high-dimensional problems with an application to a new Keynesian model," *Quantitative Economics*, 6, 1–47.

A SAMW Algorithm for Simple Two Period RBC

For any policy $\pi \in \Pi$ the value function for the simple two period RBC is

$$\begin{aligned} V^{\pi}(k_0, A_0) =& \mathbb{E}\left[\sum_{t=0}^{1} \beta^t \ln(A_t k_t^{\alpha} - \pi_t(k_t, A_t)) \middle| k_0, A_0 \right] \\ =& \mathbb{E}\left[\ln(A_0 k_0^{\alpha} - \pi_0(k_0, A_0)) + \beta \ln(A_1 k_1^{\alpha} - \pi_1(k_1, A_1)) \middle| k_0, A_0 \right] \end{aligned} \tag{31}$$

The optimal (analytic) policy $\pi^* = \{\pi_0^*, \pi_1^*\}$ is already derived as

$$\pi_0^*(k, A) = \frac{\alpha\beta}{1+\alpha\beta} A k^{\alpha}, \quad \pi_1^*(k, A) = 0 \quad \forall k, A \tag{32}$$

and plugging it into the value function above yields

$$\begin{aligned} V^*(k_0, A_0) =& \mathbb{E}\left[\ln\left(A_0 k_0^{\alpha} - \frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^{\alpha}\right) + \beta \ln\left(A_1 \left(\frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^{\alpha}\right)^{\alpha}\right) \middle| k_0, A_0 \right] \\ =& \ln\left(A_0 k_0^{\alpha} - \frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^{\alpha}\right) + \beta \mathbb{E}[\ln A_1 | k_0, A_0] + \beta \ln\left(\frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^{\alpha}\right)^{\alpha} \end{aligned} \tag{33}$$

Thus we can solve for the optimal value function given the structure of the stochastic component A_1. For example, if A follows an AR(1) process with coefficient ρ, the optimal value function would become $V^*(k_0, A_0) = \ln\left(A_0 k_0^{\alpha} - \frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^{\alpha}\right) + \beta\rho A_0 + \beta \ln\left(\frac{\alpha\beta}{1+\alpha\beta} A_0 k_0^{\alpha}\right)^{\alpha}$. An *i.i.d.* case is a special case with $\rho = 0$.

Next, let's construct the finite set Λ of 'heuristic' policy functions. First we can directly set $\pi_1 = 0$ for all $\pi = \{\pi_0, \pi_1\}$ in Λ because of the transversality condition. Then the size of the policy vector boils down to singleton, $\pi = \{\pi_0\}$ given $\pi_1 = 0$. So we can rewrite the value function with a policy $\pi \in \Lambda$ in terms of numerical programming as

$$V^{\pi}(k_0, A_0) = \ln(A_0 k_0^{\alpha} - \pi_0) + \beta \mathbb{E}[\ln A_1 | k_0, A_0] + \alpha\beta \ln \pi_0 \tag{34}$$

(34) allows us to draw the big picture for solution strategy. Through the SAMW algorithm we estimate the value $V^*(k_0, A_0)$. Then (34) with the computed value $V^*(k_0, A_0)$ on the right-hand-side becomes an equation with one unknown, π_0^*. Therefore as long as the algorithm successfully approximates the optimal value function, solving for the optimal policy function follows immediately.

Now let's describe the detailed steps for the algorithm. First we need to construct the finite 'heuristic' policy space Λ. Because this exercise is extremely simple, we can just make a grid out of the available saving choice, $[0, A_0 k_0^\alpha]$. There is one more reason for this discretizing. In general, a DSGE model does not provide a heuristic policies at initial period from economics theory perspective. Since the initial period policy is all we have for this exercise, there is no room for economics theory to roll in. In addition, the numerical example 5.1.4 of Chang et al. (2013) also constructs the policy space similarly(threshold policies), so we can safely move on with the grid as our first trial.

Algorithm A.1. ***SAMW Algorithm for Simple Two Period RBC***

0. *Scale down the maximum utility,* $\max_c(u(c)) < \frac{1}{T}$ *to bound* $0 \leq V^\pi \leq 1\ (a.s.),\ \forall \pi \in \Pi$
1. *Construct the restricted policy space* Λ *with grid of* k *elements on* $[0, A_0 k_0^\alpha]$
2. *Set* $\{\phi^1\} = \frac{1}{|\Lambda|} = \frac{1}{k}$ *for all* $\pi \in \Lambda$
3. *Draw* A_1^i *from the given model structure*
4. *For each* $\pi \in \Lambda$*, compute* $V_i^\pi(k_0, A_0)$ *from (34) replacing* $\mathbb{E}[\ln A_1 | k_0, A_0]$ *with* $\ln A_1^i | k_0, A_0$
5. *update* $\{\phi^{i+1}\}$ *with the given updating rule (21)*
6. *Iterate step 3 to 5 until* $i = 1, \ldots, N$

The above algorithm will generate the stationary distribution $\{\phi^*\}$ on Λ which concentrates on the optimal policy π^*. Step 0 stands for the updating rule (21) functions properly. This is a harmless restriction since scaling down the utility function is a positive affine transformation. It preserves preference ordering uniquely and the solution of a model as well. Several exercises turn out to be successful without normalizing the scale under the weaker condition, $\bar{u} \leq u(A_0 k_0^\alpha) = .9986$. The finite-time upper bound (24) is exceptionally problematic in that the inequality does not hold. This is due to the condition $0 \leq V^\pi \leq 1\ (a.s.),\ \forall \pi \in \Pi$ is necessary for **Theorem** 3.4 to hold.

B Euler Equation Error

It is conventional to check the accuracy of numerical solution method through calculating *Euler equation error*. For example, given the intertemporal Euler equation (28), Judd (1992)

proposed a unit-free Euler equation error as

$$EE = \frac{u_c(c_t) - \beta \sum_{s\in\mathcal{S}} p(s) u_c(c_{t+1})\{A_{t+1}(s) f_k(k_{t+1}) + (1-\delta)\}}{c^{ss}} \tag{35}$$

Where c^{ss} is the steady state consumption. However, there might be no steady state in a finite period model and the unit-free error does not have a clear interpretation yet. Den Haan and Marcet (1994) constructs unit-free error in terms of normalized consumption,

$$EE = \frac{|c_0 - u_c^{-1}(c_0)|}{u_c^{-1}(c_0)} \tag{36}$$

where $u_c^{-1}(c_0)$ is the inverse function of (28) from which the numerical solution calculated with the right-hand-side

$$u_c^{-1}(c_0) = \big[\beta \sum_{s\in\mathcal{S}} p(s) u_c(c_1(s))\{A_1(s) f_k(k_1) + (1-\delta)\}\big]^{-1} \tag{37}$$

This one has a clear, direct interpretation. For example, if the absolute value of Euler error in base 10 log is x, then the error can be measured as 10^x units of consumption goods. Using the equilibrium condition, we can compute the Euler equation error as

$$\log_{10} EE = \log_{10} |c_0 - u_c^{-1}(c_0)| - \log_{10} u_c^{-1}(c_0)$$

where $u_c^{-1}(c_0) = \big[\beta \sum_{s\in\mathcal{S}} p(s) c_1(s)^{-1} \alpha A_1(s) k_1^{\alpha-1}\big]^{-1}$. Alternately, one may also use slightly different formula under the same interpretation,

$$\log_{10} EE = \log_{10} |c_0 - u_c^{-1}(c_0)| - \log_{10} c_0$$