Timothy Hubbard
Yigit Saglam

Solution to Numerical Dynamic Programming Problems

1 Common Computational Approaches

This handout examines how to solve dynamic programming problems on a computer. In this handout we consider problems in both deterministic and stochastic environments. Nearly all of this information can be found somewhere in Judd (1998) or Rust (1996) and while we may not explicitly cite the book or paper, this entire handout essentially is applications derived from them. In this section we first discuss two possible representations of the value function - discrete approximation and smooth approximation, and then present two computational approaches, value function iteration and policy function iteration. To maintain consistency throughout this document we consider variations of the following Ramsey savings problem in each of the following sections

$$\begin{aligned} V(k,s) = \max_{c,k'} U(c) + \beta V(k',s') \\ \ni\ k' = sk^{\alpha} + (1-\delta)k - c. \\ U(c) = c^{\gamma} \end{aligned}$$

When we consider deterministic settings the shock s is irrelevant (consider $s = s' = 1$ in the above problem for these cases). Capital, k, is the state variable in each situation and our choice (control) variables are c and k', *i*.e., how much the agent consumes each period and how much she saves.

1.1 Representation of the Value Function

There are two ways we can represent the value function: discrete approximation and smooth approximation. The first step to solving a dynamic problem on a computer is to represent the continuous state variable k. To do this create a $n \times n$ matrix of points where each row of the matrix is a vector of n points the state variable can realize from $\underline{k}$ to $\overline{k}$. The i^{th} point in the vector will take on the value

$$k_i = \underline{k} + \frac{(i-1)(\overline{k} - \underline{k})}{n-1}.$$

The next step is to evaluate our utility function at each of the state variable values for each of the shocks (if in a stochastic setting). Essentially we simply evaluate the function representing current returns (current utility here) for each of the shocks s_i, and at each combination of state variable values - meaning each combination of k and k'. Note that if the utility value is negative it will not make sense, thus if this is the case we make it extremely negative (-10,000) so that the algorithm must be improved in this dimension.

This procedure must be performed whether we are going to use the discrete approximation or the smooth approximation for the value function. The difference between the two is that in a discrete approximation the value function is evaluated only at the values that we use in our capital matrix. We then use "brute force" iteration for this partitioned state space. Thus there is a trade off, a finer grid of points will yield a more accurate approximation but will require more computational effort.

The smooth approximation approach we assume the value function can be represented by a Chebyshev polynomial (any smooth function can be used here, *e*.g., splines, other polynomial series, etc.) of degree m. This approach is far more expensive for a given capital grid than the discrete approximation, however the appeal of the smooth approximation is that you can use far fewer points (a lower n) since you are interpolating the function at all points between particular nodes. The other benefit to this smooth approach is that at the end of the problem we are left with polynomial coefficients for the Chebyshev polynomial that best approximates the value function, thus we can evaluate the value function at any feasible value for the state variable quite easily, rather than being

restricted to state values that lie in the gridspace as in the discrete approximation approach. One word of caution about using a smooth approximation is that there are no theoretical restrictions that are required when using such a flexible function like a polynomial so the parameterized value function may not retain any of the properties we know the true value function has, *e.g.*, monotonicity and concavity.

In the computational algorithm the polynomial must be evaluated at a grid of n points around the steady state capital value (which can be found using Newton's method), call it k^*. A well known result from numerical analysis is that if you want to minimize the maximum error in approximating (or interpolating) a function you should not choose to evaluate the function at equidistant points, but rather you should use what are called the Chebyshev nodes. The i^{th} Chebyshev interpolation node of n total points can be computed as

$$z_i = -\cos\left(\frac{\pi(2i-1)}{2n}\right).$$

Note that since each z results from taking the cosine of a product, $z_i \in [-1,1]$ for all $i = 1,...,n$. However, we want to use capital levels k in the interval $[\underline{k},\overline{k}]$. To transform these $z \in [-1,1]$ to $k \in [\underline{k},\overline{k}]$ we can use the following transformation

$$k_i = (z_i + 1)\left(\frac{\overline{k}-\underline{k}}{2}\right) + \underline{k}.$$

We can then use these Chebyshev capital nodes to evaluate the value function - which we have chosen to represent by a Chebyshev polynomial, so the next step is to evaluate the Chebyshev polynomial at these points. Fortunately this can be done quite efficiently (and simply) by using a recursive approach. Specifically each component of the Chebyshev polynomial of degree m, call it T_i for the $i = 0,1,...,m$ component, evaluated at point k can be found by the following

$$\begin{aligned}
&T_0(k) = 1,\\
&T_1(k) = k,\\
&T_{i+1}(k) = 2kT_i(k) - T_{i-1}(k), \text{ for } i = 1,...,m.
\end{aligned}$$

The Chebyshev polynomial of degree m is simply the sum of the $m+1$ components multiplied by their appropriate coefficient. To compute the coefficients, a_i for $i = 0,...,m$ of the polynomial we can use the following formula

$$a_i = \frac{\sum_{j=1}^{n} V(k_j)T_i(z_j)}{\sum_{j=1}^{n} T_i(z_j)^2},$$

where $V(k)$ is the value function given the current capital is k. We can then approximate the value function given current capital of k by

$$\hat{V}(k) = \sum_{i=1}^{m} a_i T_i\left(\frac{2(k-\underline{k})}{\overline{k}-\underline{k}} - 1\right). \tag{1}$$

1.2 Value Function Iteration

The idea behind value function iteration is essentially to apply the contraction mapping theorem (CMT) on a computer. The algorithm is simple and guaranteed to converge by the CMT. You start by making an initial guess for the value function at each capital point (an initial guess of zero at each point for example). You compute the first iteration of the value function by considering the future value as your initial guess. This will yield a new value (the sum of the current payoff and the discounted (expected) future payoff). You can use this value as the future value in the next iteration to produce a new value, etc. Specifically the value function iteration algorithm can be described in the following steps:

1. Make an initial guess, call it $\mathbf{V}$, for the value function at each possible state, since we have n different possible values for the state variable k this will be a $n \times 1$ vector in a deterministic setting, or a $n \times s$ matrix in a stochastic setting with s possible shocks.

2. Compute $TV(k,s_i) = U(k,s_i) + \beta \sum_j \pi_{ij} V(k,s_j)$ for each shock level (if applicable) and each level of capital.
3. Determine if $||\mathbf{TV} - \mathbf{V}|| < \varepsilon$,
   (a) if so the optimal value function has been found, $\mathbf{V}^* = \mathbf{TV}$.
   (b) if not then set $\mathbf{V} = \mathbf{TV}$ and go back to step 2.

1.3 Policy Function Iteration

An alternative to value function iteration is policy function iteration. The idea is to guess an optimal policy function (assuming it's stationary) and evaluate the future value function given this policy function. Then determine the policy function that would maximize the current value function which will generate a new policy *i*mprovement. Continue iterating on the policy evaluation and improvement until the difference in the policy functions falls below a prespecified tolerance level. The following algorithm implements the policy function iteration procedure:

1. Make a guess for an initial policy call it k (*e*.g., $k(s) = \arg\max_k U(k,s)$).
2. Assuming the guess is a stationary policy, compute the implied $V(k,s)$.
3. Improvement step: improve on policy k_0 by solving
$$k' = \arg\max_k U(k,s) + \beta \sum_{s'} V(k,s')$$
4. Determine if $||k' - k|| < \varepsilon$,
   (a) if so the optimal policy function has been found, $k^* = k'$.
   (b) if not then set $k = k'$ and go back to step 2.

2 Deterministic Setting

Here we consider the dynamic programming problem in the most basic setting - a deterministic setting, meaning there are no technology shocks and you know with certainty the capital you will have tomorrow, given your consumption (and hence savings) decision today.

2.1 Problem

Solve the following dynamic programming problem numerically

$$V(k) = \max_{c,k'} c^\gamma + \beta V(k') \ni k' = k^\alpha + (1-\delta)k - c. \tag{2}$$

Specifically find the steady state level of capital using Newton's method and compute the optimal value function, policy function, and consumption path by assuming the value function can be represented by a smooth approximation approach.

2.2 Solution

Note that we can substitute the constraint into the Bellman equation in (2) to obtain the following univariate optimization problem

$$V(k) = \max_{k'} \left(k^\alpha + (1-\delta)k - k'\right)^\gamma + \beta V(k').$$

Taking a first order condition with respect to k' yields

$$-\gamma \left(k^\alpha + (1-\delta)k - k'\right)^{\gamma-1} + \beta V'(k') = 0. \tag{3}$$

By the Benviniste-Scheinkman theorem we know

$$V'(k') = \gamma\left(k'^{\alpha} + (1-\delta)k' - k''\right)^{\gamma-1}\left(\alpha k'^{\alpha-1} + (1-\delta)\right). \tag{4}$$

Substituting (4) into (3) gives

$$-\gamma\left(k^{\alpha} + (1-\delta)k - k'\right)^{\gamma-1} + \beta\gamma\left(k'^{\alpha} + (1-\delta)k' - k''\right)^{\gamma-1}\left(\alpha k'^{\alpha-1} + (1-\delta)\right) = 0.$$

Since we are looking for a steady state of the economy we know $k = k' = k''$ so this can be rewritten as

$$-\gamma\left(k^{\alpha} + (1-\delta)k - k\right)^{\gamma-1} + \beta\gamma\left(k^{\alpha} + (1-\delta)k - k\right)^{\gamma-1}\left(\alpha k^{\alpha-1} + (1-\delta)\right) = 0.$$

Now multiplying through by -1 and dividing by γ, reducing, and rearranging terms yields the following equation which we define to be $f(k)$

$$f(k) \equiv \left(k^{\alpha} - \delta k\right)^{\gamma-1}\left[1 - \beta\left(\alpha k^{\alpha-1} + (1-\delta)\right)\right]. \tag{5}$$

Our goal is to see when $f(k)$ is zero and to do this we can use Newton's method which prescribes we iterate on

$$k' = k - \frac{f(k)}{f'(k)}$$

until $|k' - k| < \varepsilon$. To use this method we need to derive $f'(k)$. Taking the derivative of (5) with respect to k gives

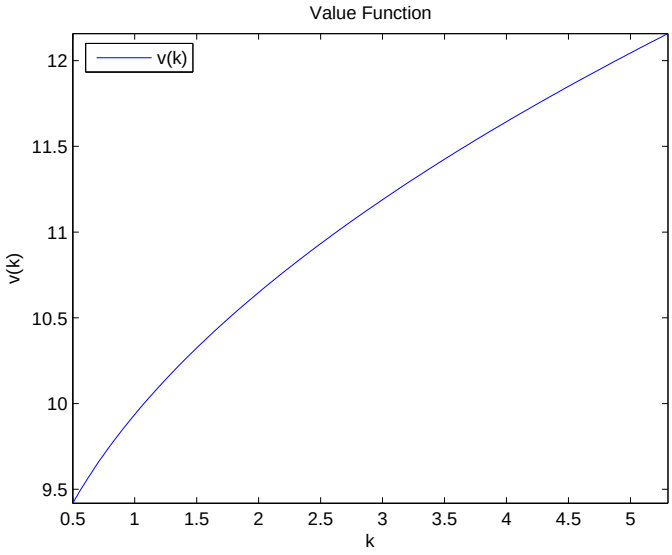
$$f'(k) = (\gamma-1)\left(k^{\alpha} - \delta k\right)^{\gamma-2}\left(\alpha k^{\alpha-1} - \delta\right)\left[1 - \beta\left(\alpha k^{\alpha-1} + (1-\delta)\right)\right] + \left(k^{\alpha} - \delta k\right)^{\gamma-1}\left[-\beta\left(\alpha(\alpha-1)k^{\alpha-2}\right)\right].$$

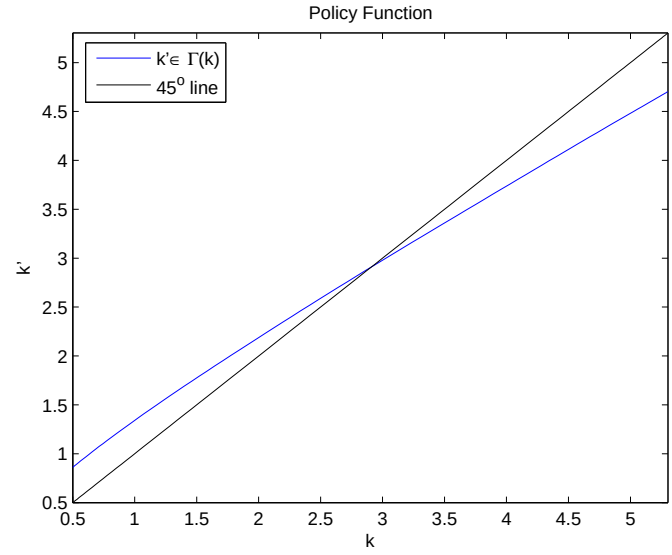
We can now implement Newton's method in Matlab using the following equation:

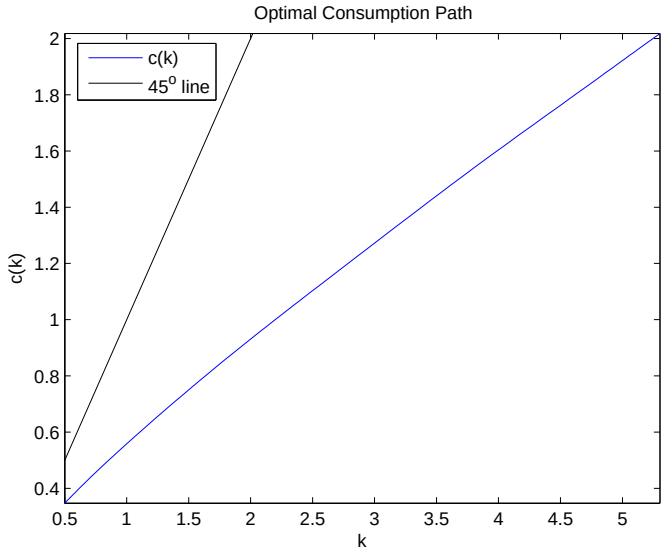
$$k' = k - \frac{\left(k^{\alpha} - \delta k\right)^{\gamma-1}\left[1 - \beta\left(\alpha k^{\alpha-1} + (1-\delta)\right)\right]}{(\gamma-1)\left(k^{\alpha} - \delta k\right)^{\gamma-2}\left(\alpha k^{\alpha-1} - \delta\right)\left[1 - \beta\left(\alpha k^{\alpha-1} + (1-\delta)\right)\right] + \left(k^{\alpha} - \delta k\right)^{\gamma-1}\left[-\beta\left(\alpha(\alpha-1)k^{\alpha-2}\right)\right]}. \tag{6}$$

To solve this problem let us assume the value function can be represented by a Chebyshev polynomial of degree m. We first need to perform the steps laid our in Subsection 1.1 to obtain a polynomial approximation to the value function, call it $\hat{V}(k)$ as in (1). The rest of the algorithm involves applying the value function iteration algorithm described in Subsection 1.2. In this algorithm each iteration yields new coefficients for the optimal value function. We stop when the Euclidean norm between the current value function and the prior value function is less than the prespecified tolerance level 1e-6. One important practical point is that each iteration the Bellman equation involves a term representing utility given the current capital level and a discounted expected future valuation term. This future valuation, represented by $V(k')$ must be evaluated at tomorrow's capital k', which means for each $V(k)$ the function $\hat{V}(\cdot)$ in (1) must be evaluated at k'.

The Matlab code for this procedure can be found in the program `dyanprog.m`. There we assume the following parameter values: $\alpha = 0.4$, $\beta = 0.9$, $\gamma = 0.5$, and $\delta = 0.1$. We approximate the policy function by an $m = 11$ degree polynomial and use $n = 200$. The program converges in about 17 seconds after 154 value function iterations. We find the steady state value of capital, $k^* = 2.9012$. The value function, policy function, and optimal consumption path for this problem are shown in figures 1, 2, and 3, respectively.


Value Function
v(k)
12
11.5
11
10.5
10
9.5
v(k)
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
k


Policy Function
k'∈ Γ(k)
45° line
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
k'
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
k



3 One Sector Growth Model with Risk

We can extend the deterministic problem to one with a stochastic environment. In this simple model consider the case where there are only two different shocks and their values and transition probabilities are known by the agents *a* priori.

3.1 Problem

Suppose there are two shocks, s_1 and s_2. Output is produced by the production function $f(k,s_i) = s_i k^\alpha$ for $i = 1,2$. Assume the same utility function, $U(c) = c^\gamma$, where we assume $\gamma = 0.5$. As before, let the discount factor be $\beta = 0.96$, $\alpha = 0.33$, $\delta = 0.1$, and assume the following transition matrix for the shock s

$$\begin{bmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Also assume that $s_1 = 2$ and $s_2 = 1$ are the values for the production shock given we are in the high or low state, respectively. Solve for the optimal decision rule and plot the optimal consumption, policy, and value functions.

3.2 Solution

We want to solve the following Bellman equation

$$V(k,s_i) = \max_{c,k'} U(c) + \beta E_{s'} V(k',s') \ni k' = s_i k^\alpha + (1-\delta)k - c,\ i = 1,2. \tag{7}$$

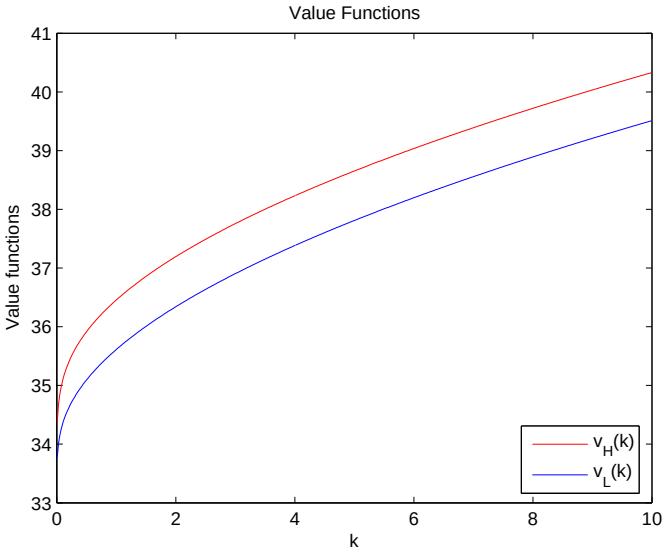
This can be rewritten as the following univariate problem

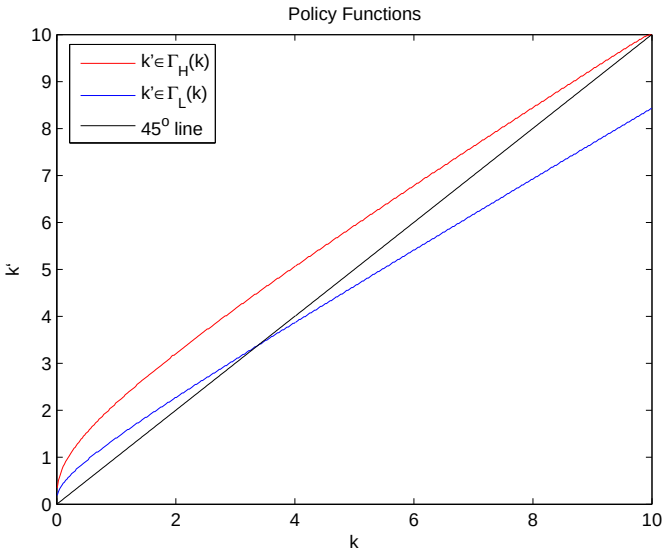
$$V(k,s_i) = \max_{k'} \left\{ \left[s_i k^\alpha + (1-\delta)k - k' \right]^\gamma + \beta \sum \pi_{ij} V(k',s'_j) \right\},\ i = 1,2 \tag{8}$$

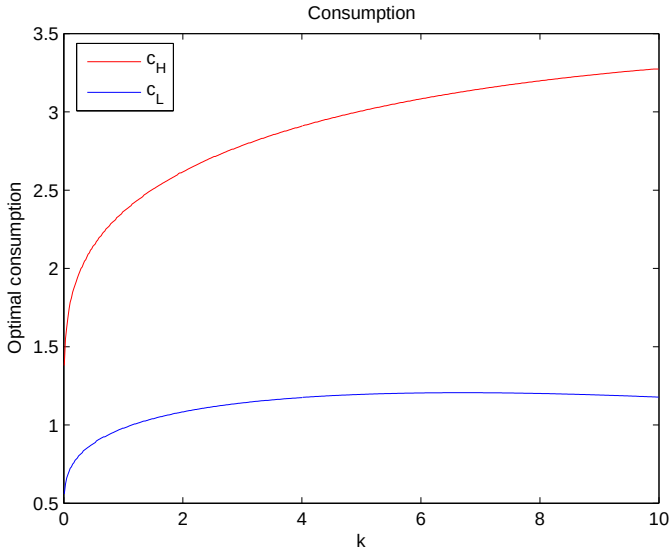
where i is the state you are currently in and π_{ij} is the probability of transitioning into state j from i.

Here we again used value function iteration. Rather than representing the value function by a Chebyshev polynomial we used a standard discrete approximation of the value function iteration to show the computational gains (time-wise) from this basic approach for a large capital space. This is reasonable if the goal of the researcher is

to simply estimate a value function, rather than evaluate it at unknown points as in the previous example. After computing the optimal value function the optimal policy function is then the sequence of capital levels that maximize the value function at each point in time, let optimal choices be denoted $k^* \in \Gamma_i(k)$ for $i = 1,2$ where $\Gamma_i(k)$ represents the feasible choices for k^* given the current level of capital is k. The optimal consumption paths can then be computed as $c_i^*(k) = s_i k^{*\alpha} + (1-\delta)k^* - k^*$. For the particular parameters specified at the beginning of the problem the optimal value functions, policy functions, and consumption paths for each shock level are given in figures 4, 5, and 6, respectively. The program converged within only 2 seconds, after 423 iterations, using $n = 200$ with an initial guess for the value function of all elements equalling zero. The Matlab code for this procedure can be found in the program `stochdptm.m`.







4 One Sector Growth Model where Shocks Follow an AR(1) Process

We can extend the simple stochastic environment discussed in the prior section by allowing for more general types of shocks. For example, in this section consider shocks that follow an AR(1) process as in Tauchen (1986).

4.1 Problem

Consider our Ramsey savings problem in a more complicated setting. Specifically let shocks follow the following AR(1) process

$$s' = \mu + \lambda s + \epsilon, \text{ with } \text{var}(\epsilon) = \sigma^2, \text{ where } |\lambda| < 1. \tag{9}$$

Here ϵ is white noise, *i.*e., ϵ is distributed with mean zero and variance σ^2. Output is produced by the production function $f(k,s) = sk^\alpha$. Assume the same utility function, $U(c) = c^\gamma$, where we assume $\gamma = 2$. As previously, let the discount factor be $\beta = 0.9$, $\alpha = 0.4$, $\delta = 0.1$. Rather than use the transition matrix previously presented, incorporate the AR(1) process for the production shock as described in (9).

4.2 Solution

We again want to solve the following Bellman equation

$$V(k,s) = \max_{c,k'} U(c) + \beta E_{s'} V(k',s') \ \ni \ k' = sk^\alpha + (1-\delta)k - c. \tag{10}$$

This can be rewritten as the following univariate problem

$$V(k,s) = \max_{k'} \left\{ \left[sk^\alpha + (1-\delta)k - k' \right]^\gamma + \beta E_{s'} V(k',s') \right\}. \tag{11}$$

The key to solving this problem on a computer is to discretize the AR(1) process. We must also assume the process stays within a bounded interval to be able to solve the problem. To do this take the approach laid out in Tauchen (1986). Specifically, Tauchen (1986) considers an AR(1) process like the one in (9), where $\mu = 0$. Denote the distribution function of ϵ as $F(\epsilon)$. We are going to approximate the continuous process in (9) by a discrete process

with N values where the shock s can take on the values $\tilde{s}_1 < ... < \tilde{s}_N$. To determine the values $\tilde{s}_i$ take on Tauchen recommends the following

$$\tilde{s}_N \equiv m\sigma_s = m\left(\frac{\sigma_\epsilon^2}{1-\lambda^2}\right)^{\frac{1}{2}},$$
$$\tilde{s}_1 = -\tilde{s}_N, \text{ and}$$
$$\tilde{s}_i = \tilde{s}_1 + \frac{(i-1)(\tilde{s}_N - \tilde{s}_1)}{N-1} \text{ (i.e., use equidistant spacing).}$$

From this we can calculate a transition matrix as we had in the previous problem. Define $p_{ij} \equiv \text{Prob}(s(t) = \tilde{s}_j | s(t-1) = \tilde{s}_i)$, this would be the element in the transition matrix row i and column j - such that $\sum_j p_{ij} = 1$. For $j \in [2, N-1]$, p_{ij} can be computed as

$$p_{ij} = F\left(\frac{\tilde{s}_j - \lambda\tilde{s}_i + w/2}{\sigma_\epsilon}\right) - F\left(\frac{\tilde{s}_j - \lambda\tilde{s}_i - w/2}{\sigma_\epsilon}\right)$$

where $w = \tilde{s}_k - \tilde{s}_{k-1}$ (note that given we are using equidistant points this value is the same for all k). These expressions can be thought of as the probability that $\lambda\tilde{s}_i + \epsilon \in [\tilde{s}_j - w/2, \tilde{s}_j + w/2]$. Otherwise the probability of transitioning from state i into state 1 is

$$p_{i1} = F\left(\frac{\tilde{s}_1 - \lambda\tilde{s}_i + w/2}{\sigma_\epsilon}\right)$$

and the probability of going from state i to state N is

$$p_{iN} = 1 - F\left(\frac{\tilde{s}_N - \lambda\tilde{s}_i - w/2}{\sigma_\epsilon}\right).$$

This discrete approximation to the conditional distribution of $s(t)$ given $s(t-1)$ will converge in the sense of probability to the true conditional distribution for the stochastic process articulated in (9).

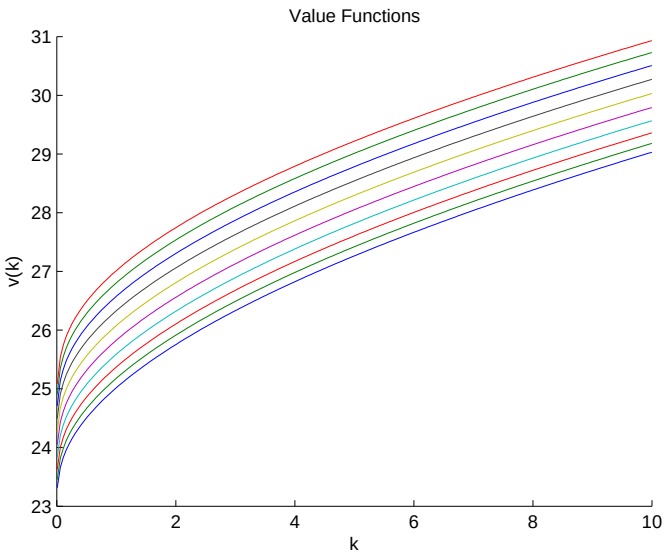
One practical concern for the above approach is how to deal with negative values for the shock. Specifically this means the firm's technology produces negative output, which does not make much economic sense. To prevent this situation we transform the shocks by letting $s = \exp(s)$ which ensures all values of the shock are positive. Another benefit of this transformation is that the grid becomes finer at the lower end and more coarse for high shock values. The rest of the program is like the value function iteration algorithm described above (using either the discrete approximation or the smooth approximation for the value function).

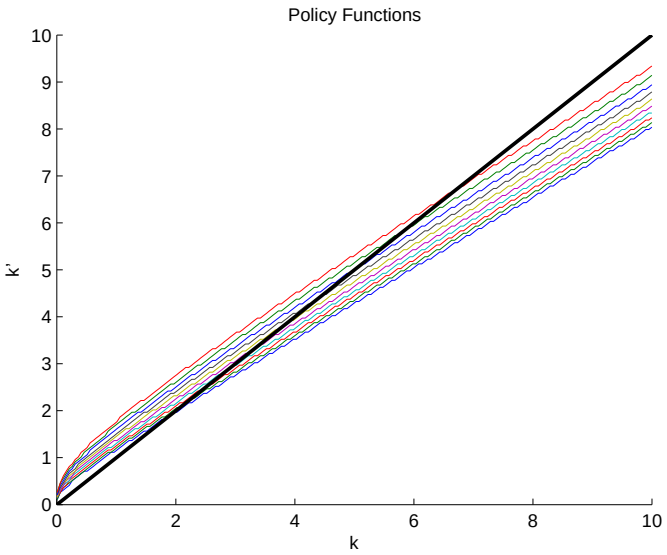
Optimal value functions policy functions, and consumption functions for each shock are given in figures 7, 8, and 9, respectively. These parameters were computed using the parameter values in Table 1. The program

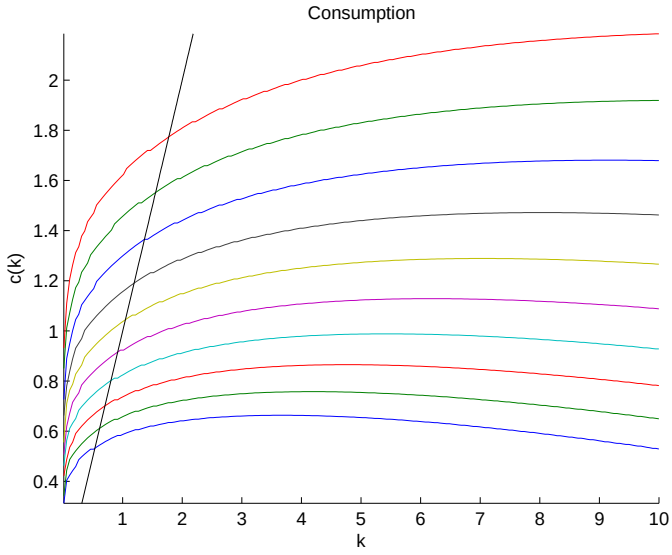
Table 1: Parameter Values for AR(1) Process Model

Parameter	Value
α	0.33
β	0.96
γ	0.5
δ	0.1
μ	0.0
λ	0.75
σ_ϵ	0.25
m	0.5

converged within 14 seconds after 434 iterations for $n = 200$ points for the capital grid and $N = 10$ possible shocks using a discrete approximation approach with an initial guess for the value function of all elements equalling zero.


Value Functions
v(k)
k
31
30
29
28
27
26
25
24
23
0
2
4
6
8
10


Policy Functions
k'
k
10
9
8
7
6
5
4
3
2
1
0
0
2
4
6
8
10



5 References

Judd, Kenneth L. (1998): *N*umerical Methods in Economics, Cambridge, MA: MIT Press.
Rust, John (1996): "Numerical Dynamic Programming in Economics," *H*andbook on Computational Economics,, vol 1, ch 14, 619–729.
Tauchen, George (1986): "Finite State Markov-Chain Approximations to Univariate and Vector Autoregressions," *E*conomics Letters, 20, 177–181.