RDCEP WORKING PAPER SERIES

The Center for Robust Decision Making
on Climate and Energy Policy

Dynamic Programming with Hermite Interpolation

Yongyang Cai and Kenneth L. Judd

Working Paper No.12-09

June, 2012

Dynamic Programming with Hermite Interpolation

Yongyang Cai

Hoover Institution, 424 Galvez Mall, Stanford University, Stanford, CA, 94305

Kenneth L. Judd*

Hoover Institution, 424 Galvez Mall, Stanford University, Stanford, CA, 94305

Abstract

Numerical dynamic programming algorithms typically use Lagrange data to approximate value functions. This paper uses Hermite data obtained from the optimization step and applies Hermite interpolation to construct approximate value functions. Several examples show that Hermite interpolation significantly improves the accuracy of value function iteration with very little extra cost.

Keywords: Numerical dynamic programming, value function iteration, Hermite interpolation

JEL Classification: C63

*The corresponding is Dr. Kenneth L. Judd, Hoover Institution, 424 Galvez Mall, Stanford University, Stanford, CA, 94305. Phone: 650-723-5866; fax: 650-723-1687. E-mail: kennethjudd@mac.com.

Email addresses: yycai@stanford.edu (Yongyang Cai), kennethjudd@mac.com (Kenneth L. Judd)

1. Introduction

All dynamic economic problems are multi-stage decision problems, often with nonlinearities that make them numerically challenging. Dynamic programming (DP) is the standard approach for dynamic optimization problems. If the state and control variables are continuous quantities, then the value function is continuous in the state and often differentiable. A numerical procedure needs to approximate the value function, but any such approximation will be imperfect since computers cannot represent the entire space of continuous functions. Many DP problems are solved by value function iteration, where the period t value function is computed from the period $t+1$ value function, and the value function is known at the terminal time T.

DP algorithms typically use Lagrange data to approximate value functions, where the Lagrange data comes from solving an optimization problem. However, that same optimization problem can also compute the slope of the value function at essentially no cost. This paper proposes using both the Lagrange data and slope information to use Hermite interpolation for constructing the value function. We explore the Hermite interpolation method in DP in detail. First, we show how to write an optimization problem so that the slope of the value function is the value of the shadow price of a constraint. Second, we demonstrate that using Hermite interpolation in DP problems with one continuous dimension improves accuracy by almost the same as if we doubled the number of points used in approximating the value function. Therefore, Hermite interpolation significantly improves value func-

tion iteration methods with very little extra cost.

This paper uses only Chebyshev polynomial approximation methods. Hermite interpolation can also be used in combination with the Schumaker shape-preserving spline method (Schumaker, 1983), or a rational function spline method (Cai and Judd, 2012a).

The paper is organized as follows. Section 2 introduces numerical DP algorithm. Section 3 describes Chebyshev interpolation and then introduces Chebyshev-Hermite interpolation. Section 4 presents the DP algorithm with Hermite interpolation. Section 5 and 6 give some numerical examples solving optimal growth problems to show the power of the DP algorithm with Hermite interpolation.

2. Numerical Methods for DP

In DP problems, if state variables and control variables are continuous quantities and value functions are continuous, then we need to approximate the value functions. We focus on using a finitely parameterizable collection of functions to approximate value functions, $V(x) \approx \hat{V}(x; \mathbf{c})$, where $\mathbf{c}$ is a vector of parameters. The functional form $\hat{V}$ may be a linear combination of polynomials, or a rational function, or a neural network function, or some other parameterization specially designed for the problem. After the functional form is chosen, we find the vector of parameters, $\mathbf{c}$, such that $\hat{V}(x; \mathbf{c})$ approximately satisfies the Bellman equation (Bellman, 1957) as accurately as possible. Value function iteration is often used to approximately solve the

Bellman equation (Judd, 1998).

The general DP problem is represented by the Bellman equation:

$$V_t(x,\theta) = \max_{a \in \mathcal{D}(x,\theta,t)} u_t(x,a) + \beta \mathbb{E}\left\{V_{t+1}(x^+,\theta^+) \mid x,\theta,a\right\}, \tag{1}$$
$$\text{s.t.} \quad x^+ = g_t(x,\theta,a,\omega),$$
$$\theta^+ = h_t(\theta,\epsilon),$$

where x is the continuous state, θ is the discrete state, $V_t(x,\theta)$ is the value function at time $t \leq T$ where $V_T(x,\theta)$ is given, a is the vector of action variables and is constrained by $a \in \mathcal{D}(x,\theta,t)$, x^+ is the next-stage continuous state with transition function g_t at time t, θ^+ is the next-stage discrete state with transition function h_t at time t, ω and ϵ are two random variables, $u_t(x,a)$ is the utility function at time t, β is the discount factor, and $\mathbb{E}\{\cdot\}$ is the expectation operator.

In the simpler case where the discrete state θ does not exist and the continuous state x is not random, the Bellman equation (1) becomes

$$V_t(x) = \max_{a \in \mathcal{D}(x,t)} u_t(x,a) + \beta V_{t+1}(x^+), \tag{2}$$
$$\text{s.t.} \quad x^+ = g_t(x,a).$$

This simpler problem can be solved by Algorithm 1 which can be naturally extended to solve the general Bellman equation (1).

Algorithm 1. *Numerical Dynamic Programming with Value Function Iteration for Finite Horizon Problems*

Initialization. *Choose the approximation nodes,* $X_t = \{x_{it} : 1 \leq i \leq m_t\}$ *for every* $t < T$*, and choose a functional form for* $\hat{V}(x; \mathbf{c})$*. Let* $\hat{V}(x; \mathbf{c}^T) \equiv V_T(x)$*. Then for* $t = T-1, T-2, \ldots, 0$*, iterate through steps 1 and 2.*

Step *1. Maximization step. Compute*

$$\begin{aligned} v_i = \max_{a_i \in \mathcal{D}(x_i,t)} \quad & u_t(x_i, a_i) + \beta \hat{V}(x_i^+; \mathbf{c}^{t+1}) \\ \text{s.t.} \quad & x_i^+ = g_t(x_i, a_i), \end{aligned}$$

for each $x_i \in X_t$*,* $1 \leq i \leq m_t$*.*

Step *2. Fitting step. Using an appropriate approximation method, compute the* $\mathbf{c}^t$ *such that* $\hat{V}(x; \mathbf{c}^t)$ *approximates* (x_i, v_i) *data.*

Algorithm 1 shows that there are two main components in value function iteration for deterministic DP problems: optimization, and approximation. In this paper we focus on approximation methods. Detailed discussion of numerical DP can be found in Cai (2009), Judd (1998) and Rust (2008).

3. Approximation

An approximation scheme consists of two parts: basis functions and approximation nodes. Approximation nodes can be chosen as uniformly spaced nodes, Chebyshev nodes, or some other specified nodes. From the viewpoint of basis functions, approximation methods can be classified as either spectral methods or finite element methods. A spectral method uses globally nonzero basis functions $\phi_j(x)$ such that $\hat{V}(x; \mathbf{c}) = \sum_{j=0}^{n} c_j \phi_j(x)$ is a degree-n

approximation. Examples of spectral methods include ordinary polynomial approximation, Chebyshev polynomial approximation, and shape-preserving Chebyshev polynomial approximation (Cai and Judd, 2012b). In contrast, a finite element method uses locally basis functions $\phi_j(x)$ that are nonzero over sub-domains of the approximation domain. Examples of finite element methods include piecewise linear interpolation, Schumaker interpolation, shape-preserving rational function spline Hermite interpolation (Cai and Judd, 2012a), cubic splines, and B-splines. See Cai (2009), Cai and Judd (2010), and Judd (1998) for more details.

3.1. Chebyshev Polynomials and Interpolation

Chebyshev basis polynomials on $[-1, 1]$ are defined as $T_j(z) = \cos(j \cos^{-1}(z))$, while general Chebyshev basis polynomials on $[a, b]$ are defined as $T_j((2x - a - b)/(b - a))$ for $j = 0, 1, 2, \ldots$. These polynomials are orthogonal under the weighted inner product: $\langle f, g\rangle = \int_a^b f(x)g(x)w(x)dx$ with the weighting function $w(x) = \left(1 - ((2x - a - b)/(b - a))^2\right)^{-1/2}$. A degree n Chebyshev polynomial approximation for $V(x)$ on $[a, b]$ is

$$\hat{V}(x; c) = \frac{1}{2}c_0 + \sum_{j=1}^{n} c_j T_j\left(\frac{2x - a - b}{b - a}\right), \tag{3}$$

where c_j are the Chebyshev coefficients.

If we choose the Chebyshev nodes on $[a, b]$: $x_i = (z_i + 1)(b - a)/2 + a$ with $z_i = -\cos((2i - 1)\pi/(2m))$ for $i = 1, \ldots, m$, and Lagrange data $\{(x_i, v_i) : i = 1, \ldots, m\}$ are given (where $v_i = V(x_i)$), then the coefficients c_j in (3) can

be easily computed by the following formula,

$$c_j = \frac{2}{m}\sum_{i=1}^{m} v_i T_j(z_i), \qquad j = 0, \ldots, n. \tag{4}$$

The method is called the Chebyshev regression algorithm in Judd (1998).

When the number of Chebyshev nodes is equal to the number of Chebyshev coefficients, i.e., $m = n + 1$, then the approximation (3) with the coefficients given by (4) becomes Chebyshev polynomial interpolation (which is a Lagrange interpolation), as $\hat{V}(x_i; c) = v_i$, for $i = 1, \ldots, m$.

3.2. Expanded Chebyshev Polynomial Interpolation

It is often more stable to use the expanded Chebyshev polynomial interpolation (Cai, 2009), as the above standard Chebyshev polynomial interpolation gives poor approximation in the neighborhood of end points. That is, we use the following formula to approximate $V(x)$,

$$\hat{V}(x; c) = \frac{1}{2}c_0 + \sum_{j=1}^{n} c_j T_j\left(\frac{2x - \widetilde{a} - \widetilde{b}}{\widetilde{b} - \widetilde{a}}\right), \tag{5}$$

where $\widetilde{a} = a - \delta$ and $\widetilde{b} = b + \delta$ with $\delta = (z_1 + 1)(a - b)/(2z_1)$. Moreover, if we choose the expanded Chebyshev nodes on $[a, b]$: $x_i = (z_i + 1)(\widetilde{b} - \widetilde{a})/2 + \widetilde{a}$, then the coefficients c_j can also be calculated easily by the expanded Chebyshev regression algorithm (Cai, 2009), which is similar to (4).

3.3. Chebyshev-Hermite Interpolation

Chebyshev interpolation does not use slope information. A more efficient approach is to compute and apply the slopes to get a closer approximation.

If we have Hermite data $\{(x_i, v_i, s_i) : i = 1, \ldots, m\}$ on $[a, b]$, where $x_i = (z_i + 1)(\widetilde{b} - \widetilde{a})/2 + \widetilde{a}$ (with $z_i = -\cos((2i-1)\pi/(2m))$) are the expanded Chebyshev nodes, $v_i = V(x_i)$ and $s_i = V'(x_i)$, then the following system of $2m$ linear equations can produce coefficients for degree $2m-1$ expanded Chebyshev polynomial interpolation on the Hermite data:

$$\frac{1}{2}c_0 + \sum_{j=1}^{2m-1} c_j T_j(z_i) = v_i, \quad i = 1, \ldots, m,$$
$$\frac{2}{\widetilde{b} - \tilde{a}} \sum_{j=1}^{2m-1} c_j T_j'(z_i) = s_i, \quad i = 1, \ldots, m,$$

where $T_j(z)$ are Chebyshev basis polynomials. After the coefficients are computed from the above linear system, we can use the polynomial (5) to approximate $V(x)$ by choosing the degree $n = 2m - 1$.

4. DP with Hermite Interpolation

The maximization step in value function iteration is

$$v_i = V_t(x_i) = \max_{a_i \in \mathcal{D}(x_i, t)} u_t(x_i, a_i) + \beta V_{t+1}(x_i^+),$$
$$\text{s.t.} \quad x_i^+ = g_t(x_i, a_i),$$

for each pre-specified approximation node x_i, $i = 1, \ldots, m$. Then it uses the Lagrange data set $\{(x_i, v_i) : i = 1, \ldots, m\}$ in the fitting step to construct an approximation $\hat{V}_t(x)$ of the value function. However, we can also compute information about slope of the value function at each approximation node almost at no cost, and make the function approximation more accurate. This

slope information allows us to use Hermite interpolation and make the numerical DP algorithm more efficient and accurate.

The envelope theorem tells us how to calculate the first derivative of a function defined by a maximization operator.

Theorem 1 (Envelope Theorem). *Let*

$$\begin{aligned} H(x) &= \max_{a} \; f(x,a) \\ &\quad \text{s.t.} \;\; g(x,a) = 0, \\ &\quad\quad\;\; h(x,a) \geq 0. \end{aligned} \tag{6}$$

Suppose that $a^*(x)$ is the optimizer of (6), and that $\lambda^*(x)$ is the vector of shadow prices for the equality constraints $g(x,a) = 0$, and $\mu^*(x)$ is the vector of shadow prices of the inequality constraints $h(x,a) = 0$,. Then

$$\frac{\partial H(x)}{\partial x} = \frac{\partial f}{\partial x}(x, a^*(x)) + \lambda^*(x)^\top \frac{\partial g}{\partial x}(x, a^*(x)) + \mu^*(x)^\top \frac{\partial h}{\partial x}(x, a^*(x)). \tag{7}$$

Formula (7) expresses the value of $\partial H(x)/\partial x$ in terms of the shadow prices, objective gradient, constraints gradients at the optimum.

However, Formula (7) is often costly to evaluate. Fortunately we can rewrite the optimization problem so that solver outputs the value of $\partial H(x)/\partial x$, in a very simple and clean formula. Corollary 1 shows us how.

Corollary 1. *The optimization problem,*

$$\begin{aligned} H(x) &= \max_{a} \; f(x,a) \\ &\quad \text{s.t.} \;\; g(x,a) = 0, \\ &\quad\quad\;\; h(x,a) \geq 0, \end{aligned} \tag{8}$$

is equivalent to

$$\begin{aligned} H(x) \;=\; \max_{a,y} \;& f(y,a) \\ \text{s.t.} \;& g(y,a) = 0, \\ & h(y,a) \geq 0, \\ & x - y = 0, \end{aligned} \tag{9}$$

and

$$\frac{\partial H(x)}{\partial x} = \lambda^*(x),$$

where $\lambda^*(x)$ is the vector of shadow prices of the trivial constraint $x - y = 0$.

The corollary follows from the envelope theorem. In Corollary 1, we take the optimization problem (8), add variable y, add constraint $x - y = 0$, and replace x by y in the objective function and all the other constraints. If we use this reformulation in an optimization solver, then $\partial H(x)/\partial x$ is the vector of shadow prices of the trivial constraint $x - y = 0$ in the output of the solver. This frees us from computing the more complex formula (7) in the envelope theorem.

Using Corollary 1, Algorithm 2 shows how to efficiently use Hermite interpolation in the numerical DP algorithms.

Algorithm 2. *Numerical Dynamic Programming with Value Function Iteration and Hermite Interpolation for Finite Horizon Problems*

Initialization. *Choose the approximation nodes, $X_t = \{x_{it} : 1 \leq i \leq m_t\}$ for every $t < T$, and choose a functional form for $\hat{V}(x; \mathbf{c})$. Let*

$\hat{V}(x; \mathbf{c}^T) \equiv V_T(x)$. *Then for* $t = T-1, T-2, \ldots, 0$, *iterate through steps 1 and 2.*

Step *1. Maximization step. For each* $x_i \in X_t$, $1 \le i \le m_t$, *compute*

$$\begin{aligned} v_i \quad = \quad & \max_{a_i \in \mathcal{D}(y_i,t), y_i} \; u_t(y_i, a_i) + \beta \hat{V}(x_i^+; \mathbf{c}^{t+1}), \\ & \text{s.t.} \qquad x_i^+ = g_t(y_i, a_i), \\ & \qquad\quad\; x_i - y_i = 0, \end{aligned}$$

and

$$s_i = \lambda_i^*,$$

where λ_i^* *is the vector of shadow prices of the constraint* $x_i - y_i = 0$.

Step *2. Hermite fitting step. Using an appropriate approximation method, compute the* $\mathbf{c}^t$ *such that* $\hat{V}(x; \mathbf{c}^t)$ *approximates* (x_i, v_i, s_i) *data.*

We can easily extend the above algorithm to solve the general DP model (1).

5. Examples for Deterministic Optimal Growth Problems

A deterministic optimal growth problem is to find the optimal consumption function and the optimal labor supply function such that the total utility over the T-horizon time is maximal[1], i.e.,

$$\begin{aligned} V_0(k_0) = \max_{k_t, c_t, l_t} \quad & \sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T V_T(k_T), \qquad (10) \\ \text{s.t.} \quad & k_{t+1} = F(k_t, l_t) - c_t, \quad 0 \le t < T, \\ & \underline{k} \le k_t \le \bar{k}, \quad 1 \le t \le T, \end{aligned}$$

[1]Please see Judd (1998) for a detailed description of this.

where k_t is the capital stock at time t with k_0 given, c_t is the consumption, l_t is the labor supply, $\underline{k}$ and $\bar{k}$ are given lower and upper bound of k_t, β is the discount factor, $F(k, l)$ is the aggregate production function, $V_T(x)$ is a given terminal value function, and $u(c_t, l_t)$ is the utility function. This objective function is time-separable, so this can be modeled as a DP problem (2), and then we can use DP methods to solve it.

In the examples, the discount factor is $\beta = 0.95$, the aggregate production function is $F(k, l) = k + Ak^{\alpha}l^{1-\alpha}$ with $\alpha = 0.25$ and $A = (1 - \beta)/(\alpha\beta)$. The state range of capital stocks is set as $[0.2, 3]$, i.e., $\underline{k} = 0.2$ and $\overline{k} = 3$. The utility function is

$$u(c, l) = \frac{(c/A)^{1-\gamma} - 1}{1 - \gamma} - (1 - \alpha)\frac{l^{1+\eta} - 1}{1 + \eta}. \tag{11}$$

The functional forms for utility and production imply that the steady state of the infinite horizon deterministic optimal growth problems is $k_{ss} = 1$, and the optimal consumption and the optimal labor supply at k_{ss} are respectively $c_{ss} = A$ and $l_{ss} = 1$.

5.1. DP Solution of Deterministic Optimal Growth Problems

The DP formulation of the deterministic optimal growth problem (10) in Algorithm 2 is:

$$\begin{aligned} V_t(k) \quad = \quad & \max_{k^+,c,l,y} \quad u(c,l) + \beta V_{t+1}(k^+), \\ & \text{s.t.} \quad k^+ = y + f(y,l) - c, \\ & \qquad k - y = 0, \\ & \qquad \underline{k} \le k^+ \le \bar{k}, \end{aligned} \tag{12}$$

for $t < T$, where the terminal value function $V_T(k)$ is given. Here k is the state variable and (c, l) are the control variables, and the dummy variable y and the trivial constraint $k - y = 0$ are used in order to get the slopes of value functions directly from the optimization solver as described in Algorithm 2.

5.2. Error Analysis

We next use some basic examples to study the usefulness of Hermite interpolation for DP. More specifically, we take finite-horizon versions of the optimal growth problem, compute the "true" optimal solution on a large set of test points for initial capital $k_0 \in [\underline{k}, \bar{k}]$, and then compare those results with the computed optimal solution from numerical DP algorithms. To get the "true" optimal solution, we use nonlinear programming to solve the optimal growth model (10), for every test point of initial capital k_0. In the examples, we choose SNOPT (Gill et al., 2005) in GAMS environment (McCarl, 2011) as the optimizer.

Table 1 lists errors of optimal solutions computed by numerical DP algorithms with or without Hermite information when $T = 100$ and terminal value function $V_T(k) \equiv 0$. The computational results of numerical DP algorithms with or without Hermite information are given by our GAMS code. We use the expanded Chebyshev polynomial interpolation as the approximation method, and we use SNOPT as the optimizer in the maximization step of DP. In Table 1, on the same m expanded Chebyshev nodes, Algorithm 1 uses degree $m-1$ expanded Chebyshev polynomial interpolation on Lagrange data, while Algorithm 2 uses degree $2m-1$ expanded Chebyshev polynomial interpolation on Hermite data.

The errors for optimal consumption at time 0 are computed by

$$\max_{k\in[0.2,3]} \frac{|c^*_{0,\mathrm{DP}}(k) - c^*_0(k)|}{|c^*_0(k)|},$$

where $c^*_{0,\mathrm{DP}}$ is the optimal consumption at time 0 computed by numerical DP algorithms on the model (12), and c^*_0 is the "true" optimal consumption directly computed by nonlinear programming on the model (10). The errors for optimal labor supply at time 0, $l^*_{0,\mathrm{DP}}$, have the similar computation formula.

Table 1 shows that value function iteration with Hermite interpolation is more accurate than Lagrange interpolation, with about one or two digits accuracy improvement. For example, data line 1 in Table 1) assumes $\gamma = 0.5$, $\eta = 0.1$, and $m = 5$ approximation nodes. For this case, the error in consumption is 0.12 for Lagrange data, and drops to 0.012 when we use Hermite

Table 1: Errors of optimal solutions computed by numerical DP algorithms with expanded Chebyshev interpolation on m expanded Chebyshev nodes using Lagrange data vs. Hermite data for deterministic growth problems

			error of c_0^*		error of l_0^*	
γ	η	m	Lagrange	Hermite	Lagrange	Hermite
0.5	0.1	5	1.2(−1)	1.2(−2)	1.9(−1)	1.8(−2)
		10	6.8(−3)	3.1(−5)	9.9(−3)	4.4(−5)
		20	2.3(−5)	1.5(−6)	3.2(−5)	2.3(−6)
0.5	1	5	1.4(−1)	1.4(−2)	6.1(−2)	5.6(−3)
		10	7.7(−3)	3.7(−5)	3.1(−3)	1.6(−5)
		20	2.6(−5)	6.5(−6)	1.1(−5)	3.0(−6)
2	0.1	5	5.5(−2)	6.1(−3)	2.7(−1)	3.6(−2)
		10	3.5(−3)	2.1(−5)	2.0(−2)	1.2(−4)
		20	1.6(−5)	1.4(−6)	9.1(−5)	7.6(−6)
2	1	5	9.4(−2)	1.1(−2)	1.3(−1)	1.7(−2)
		10	5.7(−3)	3.9(−5)	9.2(−3)	6.1(−5)
		20	2.8(−5)	4.7(−6)	4.3(−5)	8.0(−6)
8	0.1	5	2.0(−2)	2.2(−3)	3.6(−1)	4.9(−2)
		10	1.2(−3)	8.5(−6)	2.7(−2)	1.9(−4)
		20	6.1(−6)	1.0(−6)	1.4(−4)	4.4(−6)
8	1	5	6.6(−2)	7.2(−3)	3.4(−1)	4.5(−2)
		10	3.0(−3)	2.6(−5)	2.0(−2)	1.7(−4)
		20	2.0(−5)	0.0(−7)	1.3(−4)	2.1(−7)

Note: $a(k)$ means $a \times 10^k$.

interpolation. Similarly the error in labor supply is 0.19 for Lagrange data, and drops to 0.018 when we use Hermite interpolation. For both consumption and labor supply, Hermite interpolation is about 10 times more accurate than Lagrange interpolation using 5 approximation nodes. The next line chooses $m = 10$ approximation nodes, and in this case both consumption and labor supply errors drop by a factor about 200 when we switch from Lagrange to Hermite interpolation. The third line in Table 1 chooses $m = 20$. In this case, the switch reduces errors by a factor of about 15.

The rest of Table 1 examines different γ and η, and shows the same patterns for the reduction of errors when we switch from Lagrange to Hermite interpolation. Table 1 assumes $T = 100$ and $V_T(k) \equiv 0$. We also have similar results for different $T = 2, 3, 5, 10, 50$ and/or other terminal value functions $V_T(k) = u(F(k, 1) - k, 1)/(1 - \beta)$ and $V_T(k) = -100(k - 1)^2$. Moreover, we find that when T increases, the errors do not accumulate. This follows that the backward value function iterations are stable for these examples. The approximated value functions have similar accuracy results.

Since the slope information of value functions can be obtained almost freely in computation cost, Algorithm 2 has almost twice efficiency of Algorithm 1. The computation times of both numerical DP algorithms also show that they are almost proportional to number of nodes, i.e., number of optimization problems in the maximization step of numerical DP algorithm, regardless of approximation using Lagrange or Hermite data.

6. Examples for Stochastic Optimal Growth Problems

When the capital stock is dependent on a random economic shock θ, the optimal growth problem (10) becomes a stochastic dynamic optimization problem,

$$V_0(k_0, \theta_0) = \max_{k_t, c_t, l_t} \quad \mathbb{E}\left\{\sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T V_T(k_T, \theta_T)\right\}, \tag{13}$$
$$\text{s.t.} \quad k_{t+1} = F(k_t, l_t, \theta_t) - c_t, \quad 0 \le t < T,$$
$$\theta_{t+1} = h(\theta_t, \epsilon_t), \quad 0 \le t < T,$$
$$\underline{k} \le k_t \le \bar{k}, \quad 1 \le t \le T,$$

where θ_t is a discrete time process with its transition function h, ϵ_t is a serially uncorrelated random process, and $F(k, l, \theta)$ is the aggregate production function dependent on the economic shock.

In the examples, the discount factor is $\beta = 0.95$, the aggregate production function is $F(k, l, \theta) = k + \theta A k^\alpha l^{1-\alpha}$ with $\alpha = 0.25$ and $A = (1 - \beta)/(\alpha\beta)$, and the utility function is the same with (11). The terminal value function is $V_T(k, \theta) = u(F(k, 1, 1) - k, 1)/(1 - \beta)$. The range of capital stocks is $[0.2, 3]$, i.e., $\underline{k} = 0.2$ and $\overline{k} = 3$. We assume that θ_t is a Markov chain, the possible values of θ_t are $\vartheta_1 = 0.9$ and $\vartheta_2 = 1.1$, and the probability transition matrix from θ_t to θ_{t+1} is

$$\begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{bmatrix},$$

for $t = 0, \ldots, T - 1$.

6.1. DP Solution of Stochastic Optimal Growth Problems

The DP formulation of the stochastic optimal growth problem (13) in stochastic extension of Algorithm 2 is:

$$\begin{aligned} V_t(k,\theta) \;=\; & \max_{k^+,c,l,y} \; u(c,l) + \beta \mathbb{E}\left\{V_{t+1}(k^+,\theta^+)\right\}, \\ & \text{s.t.} \quad k^+ = F(y,l,\theta) - c, \\ & \qquad \theta^+ = h(\theta,\epsilon), \\ & \qquad k - y = 0, \\ & \qquad \underline{k} \le k^+ \le \bar{k}, \end{aligned} \tag{14}$$

for $t < T$, where k is the continuous state, θ is the discrete state, k^+ and θ^+ are next-stage continuous and discrete states respectively, ϵ is a random variable, and the terminal value function $V_T(k,\theta)$ is given.

6.2. Tree Method

To solve the stochastic model (13) using nonlinear programming, we can apply a tree method that is a generalized approach from solving the deterministic model (10) by nonlinear programming. Assume that θ_t is a Markov chain with possible states, $\vartheta_1, \ldots, \vartheta_m$, and the probability of going from state ϑ_i to state ϑ_j in one step is

$$\mathbb{P}(\theta_{t+1} = \vartheta_j \,|\, \theta_t = \vartheta_i) = q_{i,j}.$$

Therefore, from a given initial state at time 0 to a state at time t, there are m^t paths of θ_t, for $0 \le t \le T$. Since the next-stage capital is only dependent

on the current-stage state and control, there are only m^{t-1} paths of optimal capital, from a given initial state at time 0 to a state at time t, for $1 \leq t \leq T$. In a mathematical formula, given an initial state (k_0, θ_0), the capital at path n and time $t+1$ is

$$k_{t+1,n} = F\left(k_{t,[(n-1)/m]+1}, l_{t,n}, \vartheta_{\mathrm{mod}(n-1,m)+1}\right) - c_{t,n},$$

for $1 \leq n \leq m^t$, where $\mathrm{mod}(n-1, m)$ is the remainder of division of $(n-1)$ by m, and $[(n-1)/m]$ is the quotient of division of $(n-1)$ by m.

In the tree method, the goal is to choose optimal consumption $c_{t,n}$ and labor supply $l_{t,n}$ to maximize the expected total utility, i.e.,

$$\begin{aligned} \max_{c_{t,n}, l_{t,n}} \quad & \sum_{t=0}^{T-1} \beta^t \sum_{n=1}^{m^t} (P_{t,n} u(c_{t,n}, l_{t,n})) + \\ & \beta^T \sum_{n=1}^{m^{T-1}} P_{T-1,n} \sum_{j=1}^{m} q_{\mathrm{mod}(n-1,m)+1,j} V_T(k_{T,n}, \vartheta_j), \end{aligned} \tag{15}$$

where $P_{t,n}$ is the probability of path n from time 0 to time t with the following recursive formula:

$$P_{t+1,(n-1)m+j} = P_{t,n} \cdot q_{\mathrm{mod}(n-1,m)+1,j},$$

for $j = 1, \ldots, m$, $n = 1, \ldots, m^t$ and $t = 1, \ldots, T-2$, where $P_{0,1} = 1$ and $P_{1,j} = \mathbb{P}(\theta_1 = \vartheta_j \,|\, \theta_0)$ for a given θ_0.

It usually becomes infeasible for a nonlinear programming optimization package to solve the stochastic problem (15) with high accuracy when $T > 10$ (but numerical DP algorithms can still solve it well), see Cai (2009). However,

in our examples, we have $m = 2$, so if we let $T = 5$, the tree method will have an accurate optimal solution as the "true" solution which can be used for error analysis of numerical DP algorithms.

6.3. Error Analysis

We examine the errors for the stochastic model in the same manner we did for the deterministic optimal growth problems: We apply nonlinear programming to get the "true" optimal solution on the model (15) for every test point of initial capital $k_0 \in [\underline{k}, \bar{k}]$ and every possible initial discrete state θ_0, and then use them to check the accuracy of the computed optimal solution from numerical DP algorithms on the model (14).

Table 2 lists errors of optimal solutions computed by numerical DP algorithms with or without Hermite information when $T = 5$. The computational results of numerical DP algorithms with or without Hermite information are given by our GAMS code, where we use degree $2m-1$ or $m-1$, respectively, expanded Chebyshev polynomial interpolation on m expanded Chebyshev nodes. And in the maximization step of DP, we use SNOPT.

The errors for optimal consumptions at time 0 are computed by

$$\max_{k\in[0.2,3],\theta\in\{0.9,1.1\}} \frac{|c^*_{0,\mathrm{DP}}(k,\theta) - c^*_0(k,\theta)|}{|c^*_0(k,\theta)|},$$

where $c^*_{0,\mathrm{DP}}$ is the optimal consumption at time 0 computed by numerical DP algorithms on the model (14), and c^*_0 is the "true" optimal consumption computed by nonlinear programming on the model (15). The similar formula applies to compute errors for optimal labor supply at time 0.

From Table 2, we can also see the similar pattern shown in Table 1. That is, value function iteration with Hermite interpolation is more accurate than Lagrange interpolation, with about one to two digit accuracy improvement. For example, data line 1 in Table 1) assumes $\gamma = 0.5$, $\eta = 0.1$, and $m = 5$ approximation nodes. For this case, the error in consumption is 0.11 for Lagrange data, and drops to 0.013 when we use Hermite interpolation. Similarly the error in labor supply is 0.19 for Lagrange data, and drops to 0.018 when we use Hermite interpolation. For both consumption and labor supply, Hermite interpolation is about 10 times more accurate than Lagrange interpolation using 5 approximation nodes. The next line chooses $m = 10$ approximation nodes, and in this case both consumption and labor supply errors drop by a factor about 200 when we switch from Lagrange to Hermite interpolation.

7. Conclusion

The paper has shown that the slope information of the value functions can be obtained almost freely, and we use that information for Hermite interpolation of the value function. The paper has applied the numerical DP algorithm with Hermite interpolation to solve optimal growth problems and then has shown that the DP method with Hermite interpolation is more accurate and efficient than the one without Hermite interpolation.

Table 2: Errors of optimal solutions computed by numerical DP algorithms with expanded Chebyshev interpolation on m expanded Chebyshev nodes using Lagrange data vs. Hermite data for stochastic growth problems

			error of c_0^*		error of l_0^*	
γ	η	m	Lagrange	Hermite	Lagrange	Hermite
0.5	0.1	5	1.1(−1)	1.3(−2)	1.9(−1)	1.8(−2)
		10	5.4(−3)	2.7(−5)	7.8(−3)	3.7(−5)
		20	1.8(−5)	4.0(−6)	2.4(−5)	4.9(−6)
0.5	1	5	1.5(−1)	1.8(−2)	6.5(−2)	7.0(−3)
		10	7.2(−3)	3.4(−5)	2.9(−3)	1.5(−5)
		20	2.4(−5)	4.9(−6)	1.1(−5)	5.0(−6)
2	0.1	5	4.9(−2)	5.0(−3)	2.5(−1)	2.8(−2)
		10	2.5(−3)	1.6(−5)	1.5(−2)	8.0(−5)
		20	1.1(−5)	3.3(−6)	5.2(−5)	4.7(−6)
2	1	5	9.1(−2)	9.7(−3)	1.3(−1)	1.5(−2)
		10	4.2(−3)	2.7(−5)	6.7(−3)	4.7(−5)
		20	1.8(−5)	3.2(−6)	3.1(−5)	5.0(−6)
8	0.1	5	2.3(−2)	2.2(−3)	4.5(−1)	4.9(−2)
		10	9.5(−4)	1.2(−5)	2.2(−2)	2.6(−4)
		20	8.9(−6)	2.7(−6)	1.9(−4)	3.7(−6)
8	1	5	2.6(−1)	1.7(−2)	1.0(−0)	1.0(−1)
		10	8.4(−3)	3.8(−5)	5.2(−2)	2.4(−4)
		20	2.6(−5)	2.5(−6)	1.6(−4)	4.8(−6)

Note: $a(k)$ means $a \times 10^k$.

[1] Bellman, R., 1957. Dynamic Programming. Princeton University Press.

[2] Cai, Y., 2009. Dynamic Programming and Its Application in Economics and Finance. PhD thesis, Stanford University.

[3] Cai, Y., Judd, K.L., 2010. Stable and efficient computational methods for dynamic programming. Journal of the European Economic Association, Vol. 8, No. 2-3, 626–634.

[4] Cai, Y., Judd, K.L., 2012a. Dynamic programming with shape-preserving rational spline Hermite interpolation. Working paper.

[5] Cai, Y., Judd, K.L., 2012b. Shape-preserving dynamic programming. Working paper.

[6] Gill, P., Murray, W., Saunders, M., 2005. SNOPT: An SQP algorithm for largescale constrained optimization. SIAM Review, 47(1), 99–131.

[7] Judd, K., 1998. Numerical Methods in Economics. The MIT Press.

[8] McCarl, B., et al., 2011. McCarl GAMS User Guide. GAMS Development Corporation.

[9] Rust, J., 2008. Dynamic Programming. In: Durlauf, S.N., Blume L.E. (Eds.), New Palgrave Dictionary of Economics. Palgrave Macmillan, second edition.

[10] Schumaker, L., 1983. On Shape-Preserving Quadratic Spline Interpolation. SIAM Journal of Numerical Analysis, 20, 854–864.

About RDCEP

The Center brings together experts in economics, physical sciences, energy technologies, law, computational mathematics, statistics, and computer science to undertake a series of tightly connected research programs aimed at improving the computational models needed to evaluate climate and energy policies, and to make robust decisions based on outcomes.

RDCEP is funded by a grant from the National Science Foundation (NSF) through the Decision Making Under Uncertainty (DMUU) program.

For more information please contact us at
info-RDCEP@ci.uchicago.edu
or visit our website:
www.rdcep.org

RDCEP
Computation Institute
University of Chicago
5735 S. Ellis Ave.
Chicago, IL, 60637 USA
+1 (773) 834 1726