Dynamic Programming with Hermite Interpolation

Kenneth Judd and Yongyang Cai

May 26, 2011

1 Introduction

A conventional dynamic programming (DP) algorithm uses the Lagrange data set to construct the approximated value function $\hat{V}_t(x)$. If we want to use a Hermite interpolation method requiring slope information of $V_t(x)$, such as Schumaker interpolation method, then it seems that we have to estimate the slopes, by use of finite difference methods. But if we can get the slope information directly, then it will save computation time and make the function approximation more accurate, such that numerical DP algorithms with Hermite interpolation will be more efficient and accurate.

In this paper, we present a DP algorithm with Hermite interpolation for solving discrete-time finite horizon decision-making problems with continuous states. The paper is constructed as follows. Section 2 gives application of DP in optimal growth problems and multi-stage portfolio optimization problems. Section 3 introduces the parametric DP algorithm and describes numerical methods in the algorithm. Section 5 presents the DP algorithm with Hermite interpolation. Section 6 and Section 7 give some numerical examples for optimal growth problems and multi-stage portfolio optimization problems respectively to show the power of the DP algorithm with Hermite interpolation.

2 Application

We first illustrate our methods with a discrete-time optimal growth problem with one good and one capital stock. It 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, i.e.,

$$\begin{aligned} V_0(k_0) = \max_{c,l} \quad & \sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T u_T(k_T), \\ \text{s.t.} \quad & k_{t+1} = F(k_t, l_t) - c_t, \quad 0 \le t < T, \end{aligned} \tag{1}$$

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, β is the discount factor, $F(k,l) = k + f(k,l)$ with $f(k_t, l_t)$ the aggregate net production function, and $u(c_t, l_t)$ is the utility function. This objective function is time-separable.

The DP version of the discrete-time optimal growth problem is

$$\begin{aligned} V_t(k) = \max_{k^+,c,l} \quad & u(c,l) + \beta V_{t+1}(k^+), \\ \text{s.t.} \quad & F(k,l) - c - k^+ = 0, \end{aligned} \tag{2}$$

which is the Bellman (1957) equation. Here k is the state variable, (k^+, c, l) are the control variables, and $V_T(k) = u_T(k)$.

We also illustrate our methods with a multi-stage portfolio optimization problem. Let W_t be an amount of money planned to be invested at stage t. Assume that available assets for trading are n stocks and a bond, where the stocks have a random return vector $R = (R_1, \ldots, R_n)$ and the bond has a riskfree return R_f for each period. If $S_t = (S_{t1}, \ldots, S_{tn})^\top$ is a vector of money invested in the n risky assets at time t, then money invested in the riskless asset is $B_t = W_t - e^\top S_t$, where e is a column vector of 1s. Thus, the wealth at the next stage is

$$W_{t+1} = R_f B_t + R^\top S_t, \tag{3}$$

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

A simple multi-stage portfolio optimization problem is to find an optimal portfolio S_t at each stage t such that we have a maximal expected terminal utility, i.e.,

$$V_0(W_0) = \max_{X_t, 0 \le t < T} \; E\{u(W_T)\},$$

where W_T is the terminal wealth derived from the recursive formula (3) with a given W_0, and u is the terminal utility function, and $E\{\cdot\}$ is the expectation operator.

The DP model of this multi-stage portfolio optimization problem is

$$\begin{aligned} V_t(W) = \max_{B,S} \quad & E\{V_{t+1}(R_f B + R^\top S)\}, \\ \text{s.t.} \quad & W - B - e^\top S = 0, \end{aligned} \tag{4}$$

for $t = 0, 1, \ldots, T-1$, where W is the state variable, B and S are the control variables, and the terminal value function is $V_T(W) = u(W)$.

3 Numerical Methods for DP

In DP problems, if state variables and control variables are continuous such that value functions are also continuous, then we have to use some approximation for the value functions, since computers cannot model the entire space of continuous functions. We focus on using a finitely parameterizable collection of functions to approximate value functions, $V(x) \approx \hat{V}(x; \mathbf{b})$, where $\mathbf{b}$ is a vector of parameters. The functional form $\hat{V}$ may be a linear combination of polynomials, or it may represent a rational function or neural network representation, or it may be some other parameterization specially designed for the problem. After the functional form is fixed, we focus on finding the vector of parameters, $\mathbf{b}$, such that $\hat{V}(x; \mathbf{b})$ approximately satisfies the Bellman equation. Numerical DP with value function iteration can solve the Bellman equation approximately (see Judd (1998)).

A general DP model is based on the Bellman equation:

$$V_t(x) = \max_{a \in \mathcal{D}(x,t)} \; u_t(x,a) + \beta E\{V_{t+1}(x^+) \mid x, a\},$$
$$\text{s.t.} \quad x^+ = g(x,a),$$

where $V_t(x)$ is called the value function at stage t, $x^+ = g(x,a)$ is the next-stage state conditional on the current-stage state x and the action a, $\mathcal{D}(x,t)$ is a feasible set of a, and $u_t(x,a)$ is the utility function at time t. The following is the algorithm of parametric DP with value function iteration for finite horizon problems.

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

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

Step 1. Maximization step. Compute

$$v_i = \max_{a_i \in \mathcal{D}(x_i,t)} \; u_t(x_i,a_i) + \beta E\{\hat{V}(x_i^+; \mathbf{b}^{t+1}) \mid x_i, a_i\}$$
$$\text{s.t.} \quad x_i^+ = g(x_i, a_i),$$

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

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

There are three main components in numerical DP: optimization, approximation, and numerical integration. In the following we focus on discussing approximation and omit the introduction of optimization and numerical integration.

4 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 $\hat{V}(x) = \sum_{j=0}^{n} c_j \phi_j(x)$ is the degree-n approximation. In this section, we present examples of spectral methods such as ordinary polynomial approximation and (tensor/complete) Chebyshev polynomial approximation. 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 shape-preserving interpolation, cubic splines, and B-splines.

A linear 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 $\phi_j(x)$ and defines $\hat{V}(x;c) = \sum_{j=0}^{n} c_j \phi_j(x)$ to be the degree n approximation. Examples of spectral methods include ordinary polynomial approximation and (tensor/complete) Chebyshev polynomial approximation. 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, cubic splines, and B-splines. See Judd (1998), Cai (2009), and Cai and Judd (2010) for more details In our examples in this paper, we applied Chebyshev polynomials and Schumaker interpolation respectively.

4.1 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 $w(x) = \left(1 - \left(\frac{2x-a-b}{b-a}\right)^2\right)^{-1/2}$. The polynomials $T_j(z)$ on $[-1, 1]$ can be

recursively evaluated:

$$\begin{aligned} T_0(z) &= 1, \\ T_1(z) &= z, \\ T_{j+1}(z) &= 2zT_j(z) - T_{j-1}(z), \quad j = 1, 2, \ldots. \end{aligned}$$

A degree n Chebyshev polynomial approximation for $V(x)$ on $a, b]$ is

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

where c_j are the Chebyshev coefficients.

If we know the values of V at some specific nodes, then we can approximate V by interpolation. That is, we find a function $\hat{V}$ such that $\hat{V}(x_i; c) = V(x_i)$ at the given nodes x_i, $i = 1, \ldots, m$. Thus, we can compute the Chebyshev coefficients by solving the system of m linear equations for a general set of nodes.

But we can easily get the Chebyshev coefficients without solving a system of linear equations, if we choose the Chebyshev interpolation nodes on $[a, b]$: $x_i = (z_i + 1)(b - a)/2 + a$ where $z_i = -\cos\left(\frac{(2i-1)\pi}{2m}\right)$, $i = 1, \ldots, m$. With these Chebyshev nodes x_i and the Lagrange data set $\{(x_i, v_i) : i = 1, \ldots, m\}$, the coefficients c_j can be calculated easily by Chebyshev regression algorithm (see Judd (1998)), i.e.,

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

where $T_j(z_i)$ can be given by the following recursive formula:

$$\begin{aligned} T_0(z) &= 1, \\ T_1(z) &= z, \\ T_{j+1}(z) &= 2zT_j(z) - T_{j-1}(z), \quad j = 1, 2, \ldots, \end{aligned}$$

4.2 Chebyshev-Hermite Interpolation

Sometimes slopes of $V(x)$ are available at the given nodes $\{x_i : i = 1, \ldots, m\}$, but the above Chebyshev interpolation method does not use these slope information. A more efficient way is to apply these slopes to get a closer approximation.

If we have Hermite data $\{(x_i, v_i, s_i) : i = 1, \ldots, m\}$ on $[a, b]$, then the following system of $2m$ linear equations can produce coefficients for degree $2m - 1$

Chebyshev polynomial interpolation on the Hermite data:

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

where $z_i = \frac{2x_i - a - b}{b-a}$ $(i = 1, \ldots, m)$ are the Chebyshev nodes in $[-1, 1]$, and $T_j(z)$ are Chebyshev basis polynomials. In the system of linear equations, $T_j(z)$ can be calculated by the recursive formula in the above section, and $T_j'(z)$ has also a recursive formula:

$$\begin{aligned} T_0'(z) &= 0, \\ T_1'(z) &= 1, \\ T_{j+1}'(z) &= 2T_j(z) + 2zT_j'(z) - T_{j-1}'(z), \quad j = 1, 2, \ldots. \end{aligned}$$

4.3 A Revised Schumaker Interpolation Method

We revise the Schumaker procedure to improve its numerical stability. We first review the Schumaker interpolation as presented in Schumaker (1983) and Judd (1998), and then describe our improvement.

Let us consider the problem on a single interval $[x_1, x_2]$. The basic Hermite problem on the interval takes the data v_1, v_2, s_1, s_2, and constructs a piecewise-quadratic function $s \in C^1[x_1, x_2]$ such that

$$s(x_i) = v_i, \quad s'(x_i) = s_i, \quad i = 1, 2.$$

Here is the interpolation algorithm.

Algorithm 2. *Schumaker Shape-Preserving Interpolation*

Step 1. Compute $\delta = (v_2 - v_1)/(x_2 - x_1)$. If $(s_1 + s_2)/2 = \delta$, then

$$s(x) = v_1 + s_1(x - x_1) + \frac{(s_2 - s_1)(x - x_1)^2}{2(x_2 - x_1)},$$

and STOP.

Step 2. If $(s_1 - \delta)(s_2 - \delta) \geq 0$, set $\xi = (x_1 + x_2)/2$; Else if $|s_2 - \delta| < |s_1 - \delta|$, then let

$$\bar{\xi} = x_1 + \frac{2(x_2 - x_1)(s_2 - \delta)}{(s_2 - s_1)},$$

and let $\xi = (x_1 + \bar{\xi})/2$; Else let

$$\underline{\xi} = x_2 + \frac{2(x_2 - x_1)(s_1 - \delta)}{(s_2 - s_1)},$$

and let $\xi = (x_2 + \underline{\xi})/2$. Then

$$s(x) = \begin{cases} v_1 + s_1(x - x_1) + C_1(x - x_1)^2, & x \in [x_1, \xi], \\ A_2 + \bar{s}(x - \xi) + C_2(x - \xi)^2, & x \in [\xi, x_2], \end{cases}$$

where $C_1 = (\bar{s} - s_1)/(2a)$, $A_2 = v_1 + as_1 + a^2 C_1$, $C_2 = (s_2 - \bar{s})/(2b)$, $\bar{s} = [2(v_2 - v_1) - (as_1 + bs_2)]/(x_2 - x_1)$, $a = \xi - x_1$, and $b = x_2 - \xi$. ■

The reader can refer to Judd (1998) and Schumaker (1983) for detailed discussion.

Notice that a and b are used as denominators for C_1 and C_2. This may give rise to the overflow problem when a or b is close to 0, i.e., ξ is close to x_1 or x_2. In addition, when s_1 is close to s_2, the overflow problem will happen again in computing $\bar{\xi}$ or $\underline{\xi}$.

Here we will propose a new version of the algorithm. In this new version (Algorithm 3), we consider the computational roundoff errors more carefully to avoid the overflow problem in Algorithm 2, such that the new version is more stable and accurate. Moreover, this new version has less computation than Algorithm 2.

From the step 2 of Algorithm 2, we know that if $(s_1 - \delta)(s_2 - \delta) < 0$ and $|s_2 - \delta| < |s_1 - \delta|$ then

$$\xi = (x_1 + \bar{\xi})/2 = x_1 + \frac{(x_2 - x_1)(s_2 - \delta)}{(s_2 - s_1)},$$

else if $|s_2 - \delta| \geq |s_1 - \delta|$ then

$$\xi = (x_2 + \underline{\xi})/2 = x_2 + \frac{(x_2 - x_1)(s_1 - \delta)}{(s_2 - s_1)}.$$

Note that

$$x_1 + \frac{(x_2 - x_1)(s_2 - \delta)}{(s_2 - s_1)} = x_2 + \frac{(x_2 - x_1)(s_1 - \delta)}{(s_2 - s_1)},$$

we just need to set

$$\xi = x_1 + \frac{(x_2 - x_1)(s_2 - \delta)}{(s_2 - s_1)},$$

if $(s_1 - \delta)(s_2 - \delta) < 0$. This saves the distinction of computing ξ under the comparison between $|s_2 - \delta|$ and $|s_1 - \delta|$ in Algorithm 2. Thus,

$$a = \xi - x_1 = (s_2 - \delta)/\lambda, \quad b = x_2 - \xi = (\delta - s_1)/\lambda,$$

where $\lambda = (s_2 - s_1)/(x_2 - x_1)$. It follows that

$$\begin{aligned} \bar{s} &= \frac{2(v_2 - v_1) - (as_1 + bs_2)}{(x_2 - x_1)} \\ &= 2\delta - \frac{(s_2 - \delta)}{\lambda}\frac{s_1}{(x_2 - x_1)} - \frac{(\delta - s_1)}{\lambda}\frac{s_2}{(x_2 - x_1)} \\ &= 2\delta - \frac{(s_2 - \delta)s_1 + (\delta - s_1)s_2}{(s_2 - s_1)} = 2\delta - \delta = \delta, \end{aligned}$$

by $\delta = (v_2 - v_1)/(x_2 - x_1)$ if $(s_1 - \delta)(s_2 - \delta) < 0$.

By $C_1 = (\bar{s} - s_1)/(2a)$, we have

$$A_2 = v_1 + as_1 + a^2 C_1 = v_1 + a(s_1 + \bar{s})/2.$$

Moreover, if $(s_1 - \delta)(s_2 - \delta) \geq 0$, then from $\xi = (x_1 + x_2)/2$ we have $a = b = (x_2 - x_1)/2$, and then

$$\bar{s} = \frac{2(v_2 - v_1) - (as_1 + bs_2)}{(x_2 - x_1)} = 2\delta - \frac{s_1 + s_2}{2}.$$

From the above computational formulas, we see that numerically there are several special cases we need to worry about : $s_1 \simeq s_2$, $s_1 \approx \delta$, or $s_2 \approx \delta$. These problems will be solved in the step 2 of Algorithm 3 by replacing $(s_1 - \delta)(s_2 - \delta) \geq 0$ by $(s_1 - \delta)(s_2 - \delta) \geq -\epsilon$ for some given tolerance $\epsilon > 0$.

Algorithm 3. *Revised Schumaker Shape-Preserving Interpolation*

Step 1. Compute $\delta = (v_2 - v_1)/(x_2 - x_1)$. If $|(s_1 + s_2)/2 - \delta| < \epsilon$, then

$$s(x) = v_1 + \left(\delta + \frac{s_1 - s_2}{2}\right)(x - x_1) + \frac{(s_2 - s_1)(x - x_1)^2}{2(x_2 - x_1)},$$

and STOP.

Step 2. If $(s_1 - \delta)(s_2 - \delta) \geq -\epsilon$, set

$$\xi = (x_1 + x_2)/2, \;\; a = b = \xi - x_1, \;\; \bar{s} = 2\delta - \frac{s_1 + s_2}{2}.$$

Else let

$$\lambda = \frac{s_2 - s_1}{x_2 - x_1}, \;\; a = (s_2 - \delta)/\lambda, \;\; b = (\delta - s_1)/\lambda, \;\; \xi = x_1 + a, \;\; \bar{s} = \delta.$$

Then

$$s(x) = \begin{cases} v_1 + s_1(x - x_1) + C_1(x - x_1)^2, & x \in [x_1, \xi], \\ A_2 + \bar{s}(x - \xi) + C_2(x - \xi)^2, & x \in [\xi, x_2], \end{cases}$$

where $C_1 = (\bar{s} - s_1)/(2a)$, $A_2 = v_1 + a(s_1 + \bar{s})/2$, and $C_2 = (s_2 - \bar{s})/(2b)$. ■

This revised algorithm not only has less computation in step 2, but also is more stable and accurate than the original version by taking account of the numerical roundoff errors.

Now we consider a general interpolation problem. If we have Hermite data $\{(x_i, v_i, s_i) : i = 1, \ldots, m\}$, we then apply the interpolant algorithm to each interval to find $\xi_i \in [x_i, x_{i+1}]$. If we have Lagrange data, $\{(x_i, v_i) : i = 1, \ldots, m\}$, we must first add estimates of the slopes and then proceed as we do with Hermite data. Schumaker suggests the following formulas for estimating slopes s_1 through s_n:

$$\begin{aligned} L &= \left[(x_{i+1} - x_i)^2 + (v_{i+1} - v_i)^2\right]^{1/2}, \quad \delta_i = \frac{v_{i+1} - v_i}{x_{i+1} - x_i}, \quad i = 1, \ldots, m-1, \\ s_i &= \begin{cases} \frac{L_{i-1}\delta_{i-1} + L_i\delta_i}{L_{i-1} + L_i}, & \text{if} \quad \delta_{i-1}\delta_i > 0, \\ 0, & \text{if} \quad \delta_{i-1}\delta_i \le 0, \end{cases} \quad i = 2, \ldots, m-1, \\ s_1 &= \frac{3\delta_1 - s_2}{2}, \quad s_m = \frac{3\delta_{m-1} - s_{m-1}}{2}. \end{aligned}$$

4.4 Shape-preserving Rational Function Spline Interpolation

Here we introduce a shape-preserving rational function spline interpolation on Hermite data $\{x_i, v_i, s_i : \ i = 1, \ldots, m\}$:

$$\hat{V}(x; \mathbf{c}) = c_{i1} + c_{i2}(x - x_i) + \frac{c_{i3}c_{i4}(x - x_i)(x - x_{i+1})}{c_{i3}(x - x_i) + c_{i4}(x - x_{i+1})}, \quad \text{when } x \in [x_i, x_{i+1}],$$

where

$$\begin{aligned} c_{i1} &= v_i, \\ c_{i2} &= \frac{v_{i+1} - v_i}{x_{i+1} - x_i}, \\ c_{i3} &= s_i - c_{i2}, \\ c_{i4} &= s_{i+1} - c_{i2}, \end{aligned}$$

for $i = 1, \ldots, m-1$.

We can verify that $\hat{V}(x; \mathbf{c})$ is in $\mathcal{C}^\infty$. Moreover, when $x \in (x_i, x_{i+1})$, if the value function is increasing and concave, i.e., $s_i > c_{i2} > s_{i+1} > 0$, then from

$c_{i3} = s_i - c_{i2} > 0$ and $c_{i4} = s_{i+1} - c_{i2} < 0$, we have

$$\begin{aligned}
\hat{V}'(x;\mathbf{c}) &= c_{i2} + \frac{c_{i3}c_{i4}(c_{i3}(x-x_i)^2 + c_{i4}(x-x_{i+1})^2)}{(c_{i3}(x-x_i) + c_{i4}(x-x_{i+1}))^2} \\
&= \frac{s_{i+1}c_{i3}^2(x-x_i)^2 + s_i c_{i4}^2(x-x_{i+1})^2 + 2c_{i2}c_{i3}c_{i4}(x-x_i)(x-x_{i+1})}{(c_{i3}(x-x_i) + c_{i4}(x-x_{i+1}))^2} \\
&> 0, \\
\hat{V}''(x;\mathbf{c}) &= \frac{-2c_{i3}^2c_{i4}^2(x_{i+1}-x_i)^2}{(c_{i3}(x-x_i) + c_{i4}(x-x_{i+1}))^3} < 0.
\end{aligned}$$

That is, the rational function spline interpolation is also increasing and concave in each (x_i, x_{i+1}).

Our numerical example in Section 7 shows that the rational function spline interpolation performs very well in DP.

5 DP with Hermite Interpolation

The conventional DP algorithm uses the maximization step to compute

$$\begin{aligned}
v_i = V_t(x_i) = \max_{a_i \in \mathcal{D}(x_i,t)} \; & u_t(x_i, a_i) + \beta E\{V_{t+1}(x_i^+) \mid x_i, a_i\}, \\
\text{s.t.} \quad & x_i^+ = g(x_i, a_i),
\end{aligned}$$

for each pre-specified node x_i, $i = 1, \ldots, m$. Then it applies the Lagrange data set $\{(x_i, v_i) : i = 1, \ldots, m\}$ in the fitting step to construct the approximated value function $\hat{V}_t(x)$. If the fitting step uses a Hermite interpolation method requiring slope information at nodes x_i of $V_t(x)$, such as Schumaker interpolation method, then it seems that we have to estimate the slopes, s_i, by use of finite difference methods. But if we can get the slope information directly, then it will save computation time and make the function approximation more accurate, such that the numerical DP algorithm with Hermite interpolation will be more efficient and accurate.

The following lemma tells us how to calculate the first derivative of a function which is defined by a maximization operator.

Lemma 1. *Let*

$$\begin{aligned}
V(x) &= \max_y \; f(x,y) \\
&\text{s.t.} \quad g(x,y) = 0.
\end{aligned} \tag{5}$$

Suppose that $y^*(x)$ is the optimizer of (5), and that $\lambda^*(x)$ is the corresponding shadow price vector. Then

$$\frac{\partial V(x)}{\partial x} = \frac{\partial f}{\partial x}(x, y^*(x)) + \lambda^*(x)^\top \frac{\partial g}{\partial x}(x, y^*(x)). \tag{6}$$

Thus, it is not necessary to compute $\partial y^*(x)/\partial x$ term in order to get $\partial V(x)/\partial x$. The shadow price vector could be reported by optimization packages, so we do not need to calculate it by ourselves.

Note that some optimization packages may report $-\lambda^*(x)$ as their shadow price vector, we should adapt it in the formula to compute $\partial V(x)/\partial x$. In the above lemma, we are assuming that the Lagrange function of the model (5) is

$$\mathcal{L}(y, \lambda; x) = f(x, y) + \lambda(x)g(x, y),$$

such that $\lambda^*(x)$ is the corresponding shadow price vector, so we can use the formula 6 to compute $\partial V(x)/\partial x$. However, if one optimization package define the Lagrange function as

$$\mathcal{L}(y, \lambda; x) = f(x, y) - \lambda(x)g(x, y),$$

and still call $\lambda(x)$ as its shadow price, then we should adapt the formula 6 into the following formula:

$$\frac{\partial V(x)}{\partial x} = \frac{\partial f}{\partial x}(x, y^*(x)) - \lambda^*(x)^\top \frac{\partial g}{\partial x}(x, y^*(x)),$$

where $\lambda^*(x)$ is the corresponding shadow price defined in the optimization package.

Lemma 1 only gives the formula to compute $\partial V(x)/\partial x$ while there are only equality constraints. But an optimization model often has bound constraints for control variables and other inequality constrains. If some inequality constraints, $h(x, y) \geq 0$, are added into the model (5), one way is to simply add a slack variable s to transform the inequality constraints into equality constraints, i.e., $h(x, y) - s = 0$ with $s \geq 0$, then use the above lemma to compute $\partial V(x)/\partial x$, as the constraint $s \geq 0$ is not directly related with x. We assume that the corresponding Lagrange function is

$$\mathcal{L}(y, \lambda, \mu, \tau; x) = f(x, y) + \lambda(x)g(x, y) + \mu(x)(h(x, y) - s) + \tau(x)s,$$

where $\lambda(x)$, $\mu(x)$ and $\tau(x)$ are respectively the corresponding shadow price vectors of $g(x, y) = 0$, $h(x, y) - s = 0$ and $s \geq 0$. Then the formula to compute $\partial V(x)/\partial x$ is

$$\frac{\partial V(x)}{\partial x} = \frac{\partial f}{\partial x}(x, y^*(x)) + \lambda^*(x)^\top \frac{\partial g}{\partial x}(x, y^*(x)) + \mu^*(x)^\top \frac{\partial h}{\partial x}(x, y^*(x)).$$

In the above formula, we have to calculate the gradient of objective function and constraint functions. However, when the objective function or constraint functions are very complicated, it is not simple to get their gradients. Moreover, when there are many constraints, it may be painful to get an explicit formula to compute $\partial V(x)/\partial x$.

In fact, no matter how many constraints there are, or how much complicated the objective or constraints are, there is a direct and simple way to solve the headache and then compute the slopes, $\partial V(x)/\partial x$, in a very simple and clean formula, by only adding one trivial control variable and one trivial constraint and simply substituting x by the trivial control variable in the objective and contraints.

Theorem 1. *(Envelope theorem) For an optimization problem,*

$$\begin{aligned} V(x) &= \max_{y} \ f(x,y) \\ &\quad \text{s.t.} \ \ g(x,y)=0, \\ &\quad\quad\ \ h(x,y)\geq 0, \end{aligned}$$

we can modify it as

$$\begin{aligned} V(x) &= \max_{y,z} \ f(z,y) \\ &\quad \text{s.t.} \ \ g(z,y)=0, \\ &\quad\quad\ \ h(z,y)\geq 0, \\ &\quad\quad\ \ x-z=0, \end{aligned} \tag{7}$$

by adding a trivial control variable z and a trivial constraint $x-z=0$. Therefore,

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

where $\lambda^{*}(x)$ is the corresponding shadow price vector of the trivial constraint $x-z=0$.

The theorem is directly followed by Lemma 1. In the theorem, we assume that the Lagrange function of the model (7) is

$$\mathcal{L}(y,\lambda,\mu,\tau;x)=f(z,y)+\lambda(x)g(z,y)+\mu(x)h(z,y)+\tau(x)(x-z),$$

where $\lambda(x)$, $\mu(x)$ and $\tau(x)$ are respectively the corresponding shadow price vectors of $g(z,y)=0$, $h(z,y)\geq 0$ and $x-z=0$. Note that we only need the shadow price vector of the trivial constraint $x-z=0$ to get $\partial V(x)/\partial x$, we do not need

to know the gradients of the objective function or other constraint functions, and we also do not need to know the shadow prices of other constraints except the trivial constraint $x - z = 0$.

Thus, we can apply Lemma 1 or Envelope theorem 1 in the DP model, so that all slopes can be obtained easily and directly from the maximization step of the DP algorithm to construct an approximation method with Hermite information. We have the following numerical DP algorithm with Hermite interpolation.

Algorithm 4. *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 $\hat{V}(x; \mathbf{b})$. Let $\hat{V}(x; \mathbf{b}^T) \equiv u_T(x)$. Then for $t = T-1, T-2, \ldots, 0$, iterate through steps 1 and 2.

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

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

and

$$s_i = \lambda_i^*,$$

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

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

This is an algorithm for general problems. For specific cases, the maximization step in Algorithm 4 could have other equivalent models to compute v_i and s_i. This depends on whether slopes s_i can be computed in a simple form.

For example, for the optimal growth DP model (2), we can get the slopes of $V_t(k)$ by Lemma 1 such that

$$V_t'(k) = \lambda^*(k) F_k(k, l^*),$$

where l^* is the optimal labor supply for the given k, and $\lambda^*(k)$ is the shadow price for the constraint $F(k, l) - c - k^+ = 0$, which is given directly by optimization packages.

For the DP model (4) of the multi-stage portfolio optimization problem, we have

$$V_t'(W) = \lambda^*(W),$$

where $\lambda^*(W)$ is the shadow price for the constraint $W - B - e^\top S = 0$. If shorting stocks or borrowing bond is not allowed, i.e., $S \geq 0$ and/or $B \geq 0$, we will still have the same formula to get the slopes of $V_t(W)$ even if the constraints $S \geq 0$ and/or $B \geq 0$ are binding, because these constraints do not include any terms with W.

6 Examples for Optimal Growth Problems

This section uses several numerical examples in finite horizon optimal growth problems to illustrate the improvement of Hermite interpolation in DP.

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

6.1 Scaling Technique

A typical utility function in optimal growth problems is a power utility with the following form

$$u(c, l) = \frac{c^{1-\gamma}}{1-\gamma} - B\frac{l^{1+\eta}}{1+\eta}$$

where $B = (1 - \alpha)A^{1-\gamma}$, while the aggregate net production function is $f(k, l) = Ak^\alpha l^{1-\alpha}$ with $A = (1 - \beta)/(\alpha\beta)$. Thus the steady state of the infinite horizon deterministic optimal growth problems is $k_{ss} = 1$ while the optimal consumption and the optimal labor supply at k_{ss} are respectively $c_{ss} = A$ and $l_{ss} = 1$.

However, when β is close to 1 and γ is large, A will be small and the optimal consumption at various levels of capital k will also be small, such that the utility function will have a large magnitude. We know a large magnitude in computation will often incur numerical errors in computation and optimization. An appropriate scaling will improve the computational accuracy, while the scaling does not affect the optimal solutions.

In the following finite horizon optimal growth examples, we will choose a scaled power utility function

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

We know this scaling will have the same optimal solutions for consumption and labor supply.

6.2 Solve Exactly with Large-Scale Optimizers

For the finite horizon optimal growth problem (1), when T is small, we can use a good large-scale optimization package to solve the problem directly, and its solution could be better than the solution of (2) given by numerical DP algorithms because of the numerical approximation errors. But when T is large, the solution of (2) given by numerical DP algorithms is usually better than the solution of (1) given by a large-scale numerical optimization package directly. In addition, if the problem becomes stochastic, i.e., the value function form becomes $V_t(x, \theta_t)$ where θ_t is a discrete time Markov chain, then it usually becomes infeasible for an optimization package to solve the stochastic problem directly with high accuracy when $T > 10$. But numerical DP algorithms can still solve it well, see Cai (2009).

In the examples of this section, we choose to solve finite horizon deterministic optimal growth problems with $T \leq 100$, so we will use the solutions of the model (1) given by SNOPT (Gill, Murray and Saunders (2005)) in AMPL code as the "true" solutions.

The scaling technique discussed in the above section is very helpful in getting good solutions from large-scale optimizers when γ and T is large. For example, in our numerical example with $\gamma = 8$ and $T = 100$, when SNOPT is applied as the large-scale optimizer, the solution without using the scaling technique is not as good as the one using the scaling technique.

6.3 DP Solution

The computational results of numerical DP algorithms with/without Hermite information are given by our Fortran code. And in the maximization step of DP, we use NPSOL (see Gill, Murray, Saunders and Wright (1994)), a set of Fortran subroutines for minimizing a smooth function subject to linear and nonlinear constraints.

Tables 1 and 2 list errors of optimal solutions computed by numerical DP algorithms with/without Hermite information when $T = 100$ and terminal value function $u_T(k) \equiv 0$. In Table 1, on the same m Chebyshev nodes, Algorithm 1 uses degree $m - 1$ Chebyshev polynomial interpolation on Lagrange data, while Algorithm 4 uses degree $2m - 1$ Chebyshev polynomial interpolation on Hermite data. In Table 2, Algorithms 1 and 4 apply Schumaker interpolation on the same m equally spaced nodes, using Lagrange data and Hermite data respectively.

The errors for optimal consumptions at stage 0 are computed by

$$\max_{k\in[0.2,3]} \frac{|c^*_{0,DP}(k) - c^*_0(k)|}{1 + |c^*_0(k)|},$$

where $c^*_{0,DP}$ is the optimal consumption at stage 0 computed by numerical DP

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

			error of c_0^*		error of l_0^*	
γ	η	m	Lagrange	Hermite	Lagrange	Hermite
0.5	0.1	5	1.1(−1)	1.2(−2)	1.9(−1)	1.8(−2)
		10	6.8(−3)	3.6(−5)	9.9(−3)	4.8(−5)
		20	1.9(−5)	0.0(−6)	2.6(−5)	8.1(−6)
0.5	1	5	1.4(−1)	1.4(−2)	6.1(−2)	5.6(−3)
		10	7.7(−3)	4.1(−5)	3.1(−3)	2.0(−5)
		20	2.3(−5)	3.1(−6)	1.0(−5)	0.0(−6)
2	0.1	5	5.5(−2)	6.1(−3)	2.7(−1)	3.6(−2)
		10	3.5(−3)	2.3(−5)	2.0(−2)	1.3(−4)
		20	1.4(−5)	4.2(−6)	8.2(−5)	2.9(−6)
2	1	5	9.4(−2)	1.0(−2)	1.3(−1)	1.7(−2)
		10	5.7(−3)	4.1(−5)	9.2(−3)	6.4(−5)
		20	2.8(−5)	7.3(−6)	4.3(−5)	9.6(−6)
8	0.1	5	2.0(−2)	2.2(−3)	3.6(−1)	4.9(−2)
		10	1.2(−3)	9.7(−6)	2.7(−2)	1.9(−4)
		20	9.5(−6)	4.3(−6)	1.3(−4)	8.4(−6)
8	1	5	6.6(−2)	7.2(−3)	3.4(−1)	4.5(−2)
		10	3.0(−3)	2.8(−5)	2.0(−2)	1.7(−4)
		20	1.9(−5)	4.5(−6)	1.2(−4)	2.1(−6)

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

algorithms, and c_0^* is the optimal consumption directly computed by SNOPT (Gill, Murray and Saunders (2005)) in AMPL code on the model (1). The errors for optimal labor supply at stage 0, $l_{0,DP}^*$, have the similar computation formula.

From the comparison of these errors, we see that the solutions computed by Algorithm 4 using Hermite data are more accurate than those from Algorithm 1 using Lagrange data, with about one or two digits accuracy improvement. We also have similar results for different $T = 2, 3, 5, 10, 50$ and/or different terminal value functions $u_T(k) = u(f(k,1),1)/(1-\beta)$ and $u_T(k) = -100(k-1)^2$. Moreover, we found that when T increases, the errors did not accumulate. This follows that the backward value function iterations are stable for these examples. The approximated value functions have similar accuracy results. For stochastic optimal growth problems with a Markov chain θ_t as an additional state variable, we still have similar results.

Moreover, after comparing the $m = 2J$ rows under Algorithm 1 with the cor-

Table 2: Errors of optimal solutions computed by numerical DP algorithms with Schumaker interpolation on m nodes using Lagrange vs. Hermite data

			error of c_0^*		error of l_0^*	
γ	η	m	Lagrange	Hermite	Lagrange	Hermite
0.5	0.1	10	1.6(−1)	2.7(−2)	2.9(−1)	3.7(−2)
		20	6.4(−2)	2.3(−3)	9.9(−2)	3.3(−3)
		40	6.6(−3)	1.2(−3)	9.3(−3)	1.7(−3)
0.5	1	10	2.1(−1)	4.4(−2)	9.8(−2)	1.7(−2)
		20	8.6(−2)	4.5(−3)	3.7(−2)	1.8(−3)
		40	1.3(−2)	1.4(−3)	5.1(−3)	5.5(−4)
2	0.1	10	5.0(−2)	7.3(−3)	3.4(−1)	4.1(−2)
		20	1.6(−2)	7.7(−4)	9.9(−2)	4.4(−3)
		40	2.0(−3)	1.9(−4)	1.1(−2)	1.1(−3)
2	1	10	1.0(−1)	1.8(−2)	1.8(−1)	2.8(−2)
		20	4.2(−2)	2.3(−3)	7.2(−2)	3.6(−3)
		40	1.1(−2)	5.2(−4)	1.8(−2)	8.2(−4)
8	0.1	10	1.4(−2)	2.1(−3)	3.5(−1)	4.7(−2)
		20	4.1(−2)	2.0(−4)	9.8(−2)	4.5(−3)
		40	8.9(−4)	9.7(−5)	2.0(−2)	2.3(−3)
8	1	10	3.8(−2)	6.4(−3)	2.8(−1)	4.0(−2)
		20	1.4(−2)	7.6(−4)	9.8(−2)	4.9(−3)
		40	4.1(−3)	2.7(−4)	2.7(−2)	1.7(−3)

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

responding $m = J$ rows for $J = 5$ or 10 in Table 1, we found that the errors are close. That tells us that if Algorithm 1 needs $2J$ Chebyshev nodes to reach a desired accuracy, then Algorithm 4 only needs J Chebyshev nodes to reach the almost same accuracy, when both algorithms use degree $2J - 1$ Chebyshev polynomial interpolation. Similar observation can be seen in Table 2 for Schumaker quadratic spline interpolation.

Since the slope information of value functions can be obtained almost freely in computation cost, Algorithm 4 has almost twice efficiency of Algorithm 1. The computation times of both numerical DP algorithms also showed 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.

7 Multi-stage Portfolio Optimization Problems

In this section, we present a numerical example of DP with Hermite interpolation to solve multi-stage portfolio optimization problems, and show the advantages of DP with Hermite interpolation in comparison with the results given by DP without Hermite interpolation.

This numerical example assumes that there are one stock and one bond available for investment , the number of periods is $T = 6$, the bond has a risk-free return $R_f = 1.04$, and the stock has a discrete random return

$$R = \begin{cases} 0.9, & \text{with probability } 1/2, \\ 1.4, & \text{with probability } 1/2. \end{cases}$$

Let the range of initial wealth W_0 as $[0.9, 1.1]$. The terminal utility function is

$$u(W) = -(W - K)^{-1}$$

with $K = 0.2$ so that the terminal wealth should be always bigger than 0.2. Moreover, we assume that borrowing or shorting is not allowed in this example, i.e., $B_t \geq 0$ and $S_t \geq 0$ for all t.

7.1 Tree Method

In the portfolio optimization problem discussed in Section 2, if we discretize the random returns of n stocks as $R = R^{(j)} = (R_{1,j}, \ldots, R_{n,j})$ with probability q_j for $1 \leq j \leq m$, then it becomes a tree model:

$$\max \sum_{k=1}^{m^T} P_{T,k} u(W_{T,k}),$$

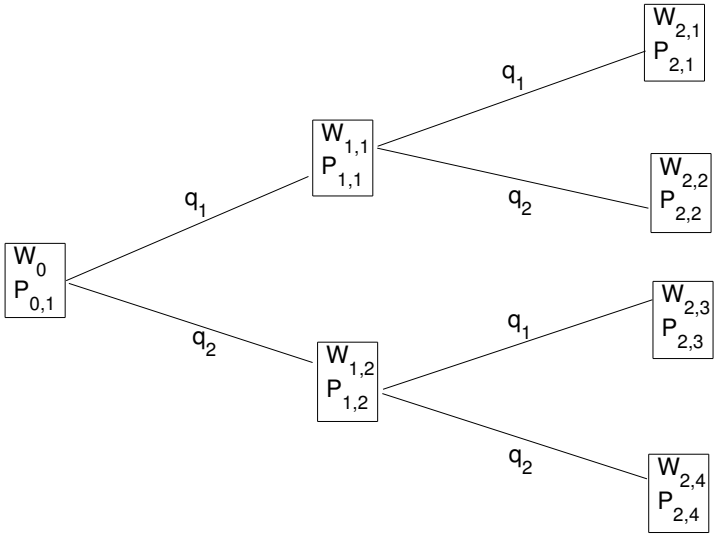

Figure 1: A binary tree with two periods

where

$$P_{t+1,k} = P_{t,[(k-1)/m]+1} q_{\mathrm{mod}(k,m)+1},$$

is the probability of scenario k at stage $t+1$, and

$$W_{t+1,k} = W_{t,[(k-1)/m]+1}(R_f B_{t,[(k-1)/m]+1} + \sum_{i=1}^{n} R_{i,\mathrm{mod}(k,m)+1} S_{i,t,[(k-1)/m]+1}),$$

is the wealth at scenario k and stage $t+1$, for $1 \le k \le m^{t+1}$ and $0 \le t < T$. Here, $W_{0,1} = W_0$ is a given initial wealth, $P_{0,1} = 1$, $\mathrm{mod}(k,m)$ is the remainder of division of k by m, and $[(k-1)/m]$ is the quotient of division of $(k-1)$ by m.

Figure 1 shows one simple tree with $m = 2$ and $T = 2$ for a portfolio with one bond and one stock ($n = 1$). The stock's random return has a probability q_1 to have a return $R_{1,1}$, and the probability $q_2 = 1 - q_1$ to have a return $R_{1,2}$. So there are two scenarios at stage 1: $(W_{1,1}, P_{1,1})$ and $(W_{1,2}, P_{1,2})$, and four scenarios at stage 2: $(W_{2,1}, P_{2,1})$, ..., $(W_{2,4}, P_{2,4})$.

The disadvantage of the tree method is that when m or T is large, the problem size will exponentially increase and it will be a big challenge for an optimizer to find an accurate solution. But the disadvantage will disappear if we use DP algorithms to solve the problem. Since the numerical example in this section is not large for the above tree model, the exact optimal allocations can be calculated by

the tree model and MINOS optimization package (Murtagh and Saunders (1978)) in AMPL code.

Figure 2 shows the optimal bond allocation B_t and stock allocation S_t, for $t = 0, 1, \ldots, 5$, computed by the tree method for the numerical example.

After obtaining these exact optimal allocations, we use them to test our numerical DP algorithms' accuracy.

7.2 DP Solutions

Since we do not allow shorting stock or borrowing bond, we should add $B \geq 0$ and $S \geq 0$ as bound constraints in the DP model of the multi-stage portfolio optimization problem.

Since the terminal utility function is $u(W) = -(W - K)^{-1}$, this implies that there should be no possibility to let the terminal wealth W_T be lower than K. It follows that the lower bound of wealth at stage t should be not less than KR_f^{t-T}. Thus, since we do not allow shorting or borrowing and R is bounded in this example, the ranges $[\underline{W}_t, \overline{W}_t]$ can be computed in an iterative way:

$$\begin{aligned} \underline{W}_{t+1} &= \max\left\{\min(R)\underline{W}_t, KR_f^{t-T}\right\} \\ \overline{W}_{t+1} &= \max(R)\bar{W}_t, \end{aligned}$$

with a given initial wealth bound $[\underline{W}_0,\ \overline{W}_0]$.

Specifically, for the numerical example with $K = 0.2$, $R_f = 1.04$, $\min(R) = 0.9$ and $\max(R) = 1.4$, after we choose $\underline{W}_0 = 0.9$ and $\overline{W}_0 = 1.1$, we have

$$\begin{aligned} [\underline{W}_1, \overline{W}_1] &= [0.81, 1.54], \\ [\underline{W}_2, \overline{W}_2] &= [0.729, 2.156], \\ [\underline{W}_3, \overline{W}_3] &= [0.656, 3.018], \\ [\underline{W}_4, \overline{W}_4] &= [0.59, 4.226], \\ [\underline{W}_5, \overline{W}_5] &= [0.531, 5.916], \\ [\underline{W}_6, \overline{W}_6] &= [0.478, 8.282]. \end{aligned}$$

We see that the range is expanding exponentially along time t. If we use a fixed range along time t in our numerical DP algorithms, then it will definitely reduce the accuracy of solutions. So here we choose the above ranges at stages $t = 0, 1, \ldots, 5$ in both the maximization step and the fitting step of Algorithm 1 or 4.

Figure 3 shows relative errors of the computed optimal stock allocations by numerical DP algorithms in comparison with the exact solutions. All the computational results of numerical DP algorithms for the example are given by MINOS

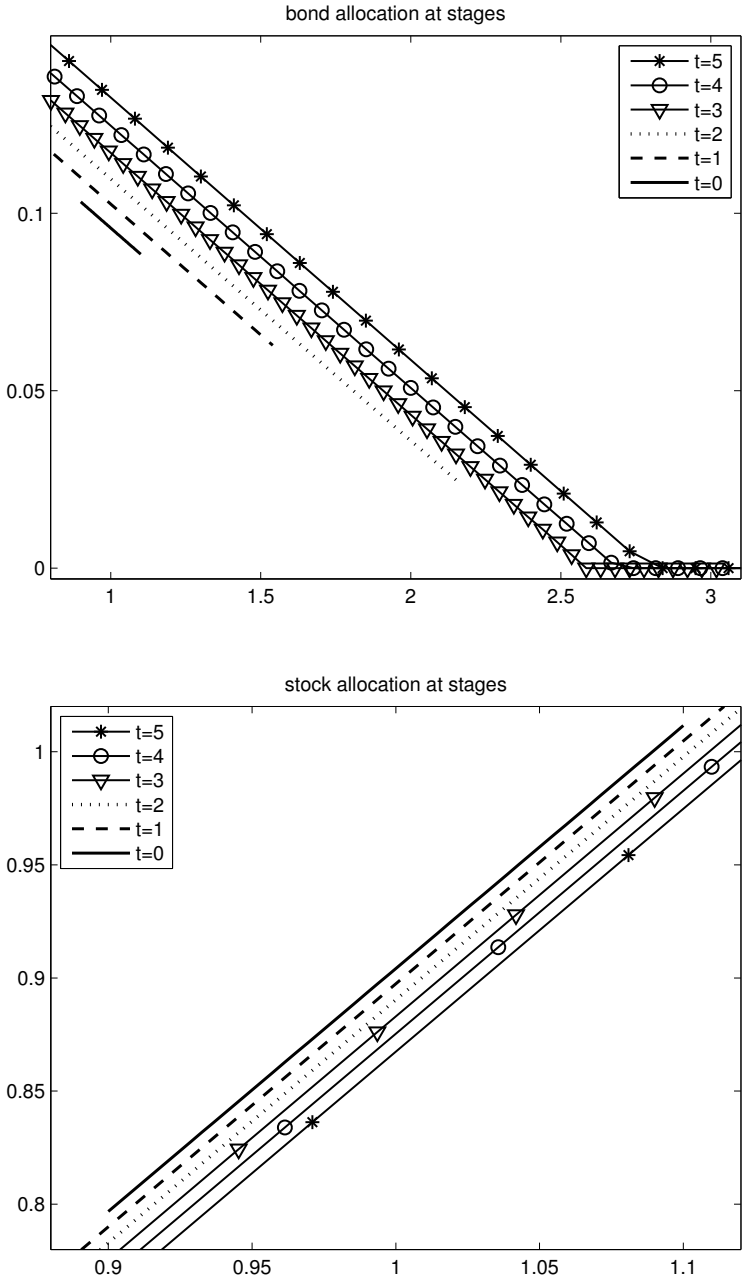

Figure 2: Exact optimal allocations

in AMPL code. And we use $m = 30$ equally-spaced nodes for each stage in the maximization step and the fitting step of Algorithm 1 or 4 in this numerical example.

The vertical axis values of plotted x-marks, squares and solid points for their corresponding horizontal axis value W_t (wealth), are given as

$$\log_{10}\left(10^{-5} + \frac{\left|S^*_{t,DP} - S^*_t\right|}{|S^*_t|}\right),$$

where S^*_t are true optimal stock allocations for the wealth W_t at stage t which are given by the tree method, and $S^*_{t,DP}$ are computed optimal stock allocation from numerical DP algorithms.

The squares are errors of solutions of Algorithm 1 with Schumaker spline interpolation using Lagrange data, the x-marks are errors of Algorithm 4 with Schumaker spline interpolation using Hermite data, and the solid points are errors of Algorithm 4 with rational function spline interpolation using Hermite data.

We see that the errors are about $O(10^{-2})$ for Schumaker interpolation using Lagrange data, while they are about $O(10^{-3})$ for Schumaker interpolation using Hermite data. So Hermite interpolation really helps to improve the accuracy of the solutions. Note that the errors become smaller when t goes to 0. This means that the errors are not accumulating at all when the number of value function iterations increases. One reason of this phenomenon is that the impact of kinks and binding constraints ($B_t = 0$) over some wealth W_t are disappearing along the backward value function iteration, which was shown in Figure 2: when $t \geq 3$, there are kinks and binding constraints for big wealths, but when $t < 3$, there are no kinks or binding constraints.

In addition, the errors of rational function spline interpolation using Hermite data keeps at about $O(10^{-5})$, so this rational function spline interpolation has the very good performance for approximating value functions with kinks.

8 Conclusion

This paper presents a numerical DP algorithm with Hermite interpolation and shows that the Hermite information in the value function approximation is very helpful in obtaining good solutions from numerical DP.

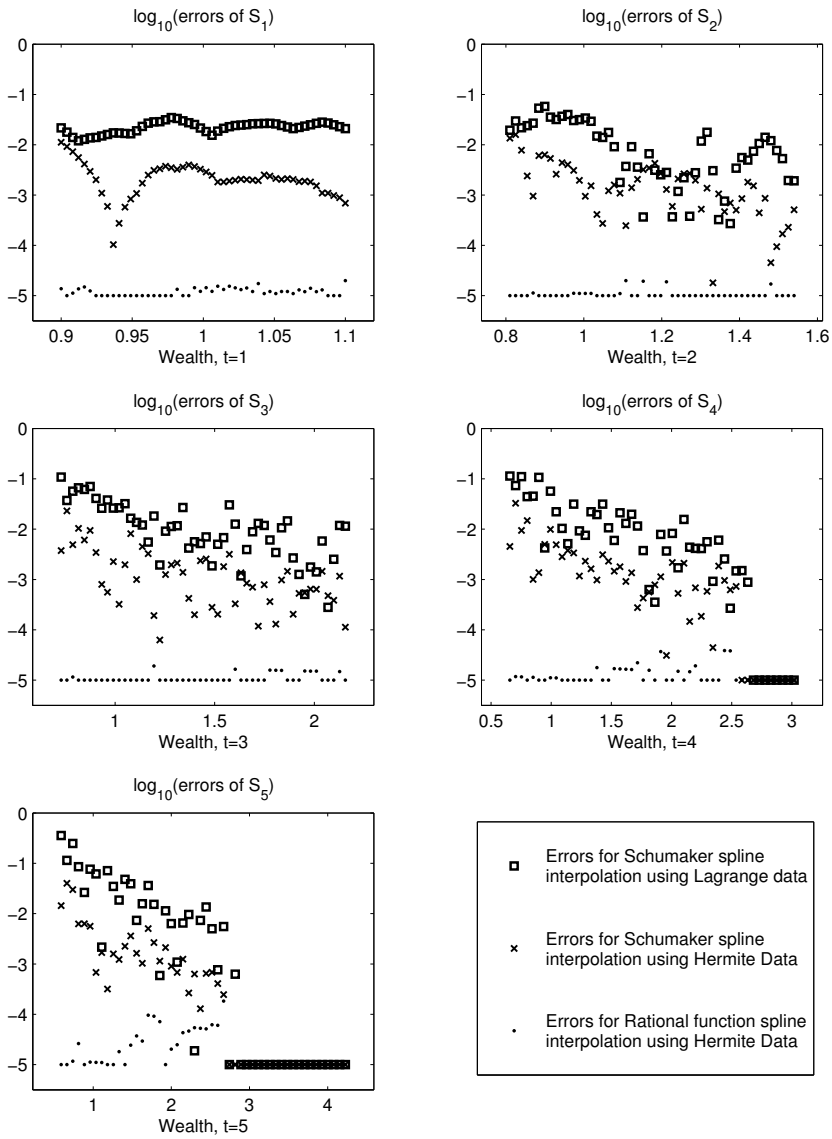

Figure 3: Relative Errors of Optimal Stock Allocations from Numerical DP

References

key-5 Bellman, Richard (1957). *Dynamic Programming*. Princeton University Press.

key-6 Cai, Yongyang (2009). *Dynamic Programming and Its Application in Economics and Finance*. PhD thesis, Stanford University.

key-7 Cai, Yongyang, and Kenneth Judd (2010). "Stable and efficient computational methods for dynamic programming". *Journal of the European Economic Association*, Vol. 8, No. 2-3, 626–634.

key-8 Gill, Philip, Walter Murray, Michael Saunders, and Margaret Wright (1994). "User's Guide for NPSOL 5.0: a Fortran Package for Nonlinear Programming". Technical report, SOL, Stanford University.

key-9 Gill, Philip, Walter Murray, and Michael Saunders (2005). "SNOPT: An SQP algorithm for largescale constrained optimization". *SIAM Review*, 47(1), 99–131.

key-10 Judd, Kenneth (1998). *Numerical Methods in Economics*. The MIT Press.

key-11 Murtagh, Bruce, and Michael Saunders (1978). "Large-scale linearly constrained optimization". *Mathematical Programming*, 14, 41–72.

key-12 Rust, John (2008). "Dynamic Programming". In: *New Palgrave Dictionary of Economics*, ed. by Steven N. Durlauf and Lawrence E. Blume. Palgrave Macmillan, second edition.

key-13 Schumaker, Larry (1983). "On Shape-Preserving Quadratic Spline Interpolation". *SIAM Journal of Numerical Analysis*, 20, 854–864.