Shape-Preserving Dynamic Programming

Kenneth Judd and Yongyang Cai

April 7, 2011

Abstract

1 Introduction

The multi-stage decision-making problems are numerically challenging. When the problems are time-separable, dynamic programming (DP) is a popular method to solve them. 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. Discretization of state variables can approximate the value functions, but it is very time consuming to get a good approximation. Polynomial or spline approximation will save much time, but computational errors may accumulate through the value function iterations. However, if the value functions are concave and their approximation preserves the concavity, then computational error accumulation problem could be solved, see Santos and Vigo-Aguiar (1998), and Maldonado and Svaiter (2001).

In this paper, we present a shape-preserving DP algorithm with value function iteration for solving the discrete-time decision-making problems with continuous states. The paper is constructed as follows. Section 2 introduces the parametric DP algorithm and describes numerical methods in the algorithm. Section 3 presents the shape-preserving DP algorithm with value function iteration. Section 4 discusses the method to compute slopes of value functions and uses the Hermite information to get a closer and better approximation. Section 5 and Section 6 give some numerical examples for optimal growth problems and multi-stage portfolio optimization problems respectively to show the power of the shape-preserving DP.

2 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:

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

where $V_t(x)$ is called the value function at stage t, x^+ is the next-stage state (may be random) 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

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

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.

A linear approximation scheme consists of two parts: basis functions and approximation nodes. 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. In our examples, we use Chebyshev polynomial interpolation, which is a spectral method. 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.

Piecewise linear interpolation is a common way applied by many scholars because of its simplicity and shape-preservation. But piecewise linear interpolation has its disadvantage: it is a challenge for optimization software to find the optimal solution in the maximization step of the numerical DP algorithm, because the approximation $\hat{V}(x)$ is only continuous but not differentiate at the nodes x_i.

Here we give a brief introduction of Chebyshev polynomials. Chebyshev basis polynomials on $[-1,1]$ are defined as $T_j(x) = \cos(j\cos^{-1}(x))$, while general Chebyshev basis polynomials on $[a,b]$ are defined as $T_j((2x-a-b)/(b-a))$ for $j = 0,1,2,\ldots$. The degree n Chebyshev polynomial approximation for $V(x)$ is

$$\hat{V}(x;c) = \sum_{0\leq|\alpha|\leq n} c_\alpha T_\alpha(x),$$

where $|\alpha|$ denotes $\sum_{i=1}^{d}\alpha_i$ for the nonnegative integer vector $\alpha = (\alpha_1,\ldots,\alpha_d)$. With a set of Chebyshev nodes x_i and the Lagrange data set $\{(x_i,v_i) : i = 1,\ldots,m\}$, the coefficients c_α can be calculated easily by Chebyshev regression algorithm

3 Shape-preserving DP

In economics and finance, many DP models have the monotone and/or concave/convex value functions such that objective functions in their optimization models preserve the shape property theoretically. So if we can have the shape-preserving value function approximation in the fitting step, then it will be very helpful to get good optimal solutions as the local optimizer will be also the global optimizer for convex optimization problems. Discretization method and piecewise linear interpolation method preserve the shape property. Another shape-preserving method is the so-called Schumaker shape-preserving interpolation method (Schumaker (1983)). A revised version of Schumaker interpolation is given in Cai (2009) and Cai and Judd (2011). Fiorot and Tabka (1991), and Steven Pruess (1993) gave some other shape-preserving splines approximation methods. Judd and Solnick (1994) discussed some theoretical properties of the shape-preserving splines in numerical DP and applied them in optimal growth problems. Wang and Judd (2000) applied a bivariate shape-preserving spline interpolation method in numerical DP to solve a savings allocation problem. Here we will extend them to a more general shape-preserving approximation method and its application in numerical DP.

3.1 Shape Correction and Preserving

For a univariate approximation problem, Schumaker interpolation method preserves the shape properties including monotonicity and concavity. But many approximation methods such as Chebyshev approximation do not have the shape-preserving property. Moreover, we also need a shape-preseving approximation method for a general multivariate problem. So we can use least-squares approximation with shape constraints to guarantee the shape-preservation. For example, if we know that the value function is strictly increasing and concave, then we could try to find the optimal parameters $\mathbf{b}$ in the following model:

$$\begin{aligned} \min_{\mathbf{b}} \quad & \sum_{i=1}^{m} \left(\hat{V}(x_i;\mathbf{b}) - v_i\right)^2 \\ \text{s.t.} \quad & \hat{V}'(y_j;\mathbf{b}) > 0, \quad j = 1,\ldots,m', \\ & \hat{V}''(y_j;\mathbf{b}) < 0, \quad j = 1,\ldots,m', \end{aligned}$$

with a sufficiently dense set of points y_j to check shape. Sometimes the shape constraints may bind, which is not desirable. In this case, we use more basis functions in the approximation until the shape constraints do not bind.

Before doing the shape-preserving fitting, we should modify the values of v_i such that the discrete set of v_i have the same properties of value function. For example, if value function is strictly increasing, then we should have $v_i < v_j$ when $x_i < x_j$, which can not be guaranteed by the optimization solvers due to the numerical errors (e.g., the optimization solver returns a local optimizer which is not global). This step is called shape correction.

3.2 Shape-preserving Chebyshev Approximation

One problem for Chebyshev approximation is the absence of shape-preservation in the algorithm. To solve this, one way is to modify the Chebyshev coefficients such that the concavity and monotonicity of the value function can be preserved at the interpolation nodes. That is, we could solve the following new least-squares problem which has a quadratic objective and linear inequality constraints:

$$\begin{aligned}
\min_{c_j} \quad & \sum_{i=1}^{m} \left(\frac{1}{2} c_0 + \sum_{j=1}^{n} c_j T_j\left(z_i\right) - v_i \right)^2 \\
\text{s.t.} \quad & \sum_{j=1}^{n} c_j T_j'\left(y_i\right) > 0, \qquad i = 1, \ldots, m', \\
& \sum_{j=1}^{n} c_j T_j''\left(y_i\right) < 0, \qquad i = 1, \ldots, m',
\end{aligned}$$

for a degree-n Chebyshev approximation, where $z_i = \frac{2x_i - a - b}{b-a}$ $(i = 1, \ldots, m)$ are the Chebyshev nodes in $[-1, 1]$, and y_i $(i = 1, \ldots, m')$ are chosen nodes in $[-1, 1]$ for shape-preserving constraints. In our numerical examples in Section 5, we choose $m' = m$ and $y_i = z_i$ for $i = 1, \ldots, m$.

We can use the following recursive formula to evaluate $T_j(y_i)$, $T_j'(y_i)$ and $T_j''(y_i)$ for $i = 1, \ldots, m'$:

$$\begin{aligned}
T_0(y) &= 1, \\
T_1(y) &= y, \\
T_{j+1}(y) &= 2yT_j(y) - T_{j-1}(y), \qquad j = 1, 2, \ldots,
\end{aligned}$$

and

$$\begin{aligned}
T_0'(y) &= 0, \\
T_1'(y) &= 1, \\
T_{j+1}'(y) &= 2T_j(y) + 2yT_j'(y) - T_{j-1}'(y), \qquad j = 1, 2, \ldots,
\end{aligned}$$

and

$$\begin{aligned}
T_0''(y) &= 0, \\
T_1''(y) &= 0, \\
T_{j+1}''(y) &= 4T_j'(y) + 2yT_j''(y) - T_{j-1}''(y), \qquad j = 1, 2, \ldots.
\end{aligned}$$

4 Shape-preserving Hermite Interpolation in DP

Cai and Judd (2011) introduces an easy and computation-free way to get slopes of value functions at nodes in DP, and then uses the Hermite data in shape-preserving Schumaker interpolation to produce a more accurate approximation such that the solution given by numerical DP methods are better than those without using slope information. The basic idea to compute slopes of value function is based on the envelope theorem discussed in the following.

4.1 Envelope Theorem

The conventional DP algorithm uses the maximization step to compute

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

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*

$$\begin{aligned} V(x) \quad = \quad \max_{y} \quad & f(x, y) \\ \text{s.t.} \quad & g(x, y) = 0. \end{aligned} \tag{1}$$

Suppose that $y^*(x)$ is the optimizer of (1), and that $\lambda^*(x)$ is the corresponding shadow price vector. Then

$$\frac{\partial V(x)}{\partial x} = \frac{\partial f}{\partial x}(x, y^*(x)) + \lambda^*(x)^\top \frac{\partial g}{\partial x}(x, y^*(x)). \tag{2}$$

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 (1) 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 2 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 2 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 (1), 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,*

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

we can modify it as

$$\begin{aligned} V(x) &= \max_{y,z} \ f(z,y) \\ & \quad \text{s.t.} \ \ g(z,y)=0, \\ & \qquad\quad\ h(z,y)\geq 0, \\ & \qquad\quad\ x-z=0, \end{aligned} \tag{3}$$

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 (3) 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.

4.2 Shape-preserving Rational Function Spline Interpolation

The slope information can be used in the shape-preserving Schumaker interpolation or Chebyshev interpolation with Hermite information, so that numerical DP methods with Hermite interpolation is more efficient and more accurate than those without Hermite interpolation, see Cai and Judd (2011). Here we introduce a shape-preserving rational function spline interpolation on Hermite data $\{x_i, v_i, s_i : \ i = 1, \ldots, m\}$:

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

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 $\hat{V}(x; c)$ 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

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

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

Our numerical examples show that the rational function spline interpolation performs very well in DP.

5 Examples for Optimal Growth Problems

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.,

$$
\begin{aligned}
V_0(k_0) = \max_{c,l} \quad & \sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T u_T(k_T), \\
\text{s.t.} \quad & k_{t+1} = F(k_t, l_t) - c_t, \quad 0 \leq t < T,
\end{aligned} \tag{4}
$$

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, and T could be finite or infinite. This objective function is time-separable.

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

$$
V_t(k) = \max_{c,l} \quad u(c, l) + \beta V_{t+1}(F(k, l) - c), \tag{5}
$$

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

In the infinite horizon optimal growth problem 5 ($T = \infty$), there is a steady state k_{ss} and its corresponding optimal control variables (c_{ss}, l_{ss}) satisfying the following equations:

$$
\begin{cases}
k_{ss} = F(k_{ss}, l_{ss}) - c_{ss}, \\
u_l(c_{ss}, l_{ss}) + u_c(c_{ss}, l_{ss}) F_l(k_{ss}, l_{ss}) = 0, \\
\beta F_k(k_{ss}, l_{ss}) = 1.
\end{cases} \tag{6}
$$

By solving the above equations, we can get the exact solution and value at the steady states, but not at other states. We will use them as indicators for accuracy of numerical DP algorithms for the infinite horizon optimal growth problem.

We use two numerical examples of the infinite horizon optimal growth model to illustrate the importance of the shape-preserving property. In the following examples, we let $\alpha = 0.25$, $\beta = 0.9$, $A = (1 - \beta)/(\alpha\beta) = 4/9$, $u(c, l) = c^{1-\gamma}/(1 - \gamma) - Bl^{1+\eta}/(1 + \eta)$ with $B = (1 - \alpha)A^{1-\gamma}$. According to the system of equations (6), we know that the steady state is $k^* = 1$ while the corresponding optimal labor is $l^* = 1$ and optimal consumption is $c^* = f(k^*, l^*) = 0.444$. Moreover, $V(k^*) = u(c^*, l^*)/(1 - \beta)$.

5.1 Example 1

In the first example, we let $\gamma = 4$ and $\eta = 1$, then the value at the steady state is $V(k^*) = -80.6836$.

Let the range of k be chosen as $[0.1, 2]$. We use the value function iteration method. The initial guess of the value function is 0 everywhere. The stopping rule for the infinite horizon variant of numerical DP algorithm is

$$\max_k \frac{|\hat{V}_{n+1}(k) - \hat{V}_n(k)|}{1 + |\hat{V}_n(k)|} < 10^{-6}.$$

The optimization solver is chosen as KNITRO (see Byrd, Nocedal and Waltz (2006)) and the program code is written in AMPL. The approximation method is chosen as the degree-40 Chebyshev polynomial interpolation method with 41 Chebyshev nodes.

At first, we try the standard Chebyshev polynomial interpolation method without shape constraints. After 634 iterations, the value function iteration method converges. When the capital state is $k^* = 1$, the corresponding computed optimal $\hat{l}^* = 3.08035$, $\hat{c}^* = 0.241709$ and $\hat{V}(1) = -74.0949$. All of these are far away from the exact optimal labor, consumption and value at the steady state. The approximated optimal control functions and value functions are shown with dashed lines in Figure 1. We see that all three functions have many large wiggles.

Next, we try the shape-preserving Chebyshev polynomial approximation method with positive gradient and negative Hessian constraints. The value function iteration method converges at the 78-th step. When the capital state is $k^* = 1$, the corresponding computed optimal $\hat{l}^* = 0.999998$, $\hat{c}^* = 0.444445$ and $\hat{V}(1) = -80.6836$. All of these are very close to the exact optimal labor, consumption and value at the steady state. The approximated optimal control functions and value functions are shown with solid lines in Figure 1. We see that all three functions are smooth, monotone and concave.

5.2 Example 2

In this example, we let $\gamma = 7$ and $\eta = 1$, then the value at the steady state is $V(k^*) = -702.7927$.

Let the range of k be chosen as $[0.1, 10]$. We use the value function iteration method. The initial guess of the value function is 0 everywhere. The stopping rule for the DP method is $\|V_{n+1} - V_n\| < 10^{-6}$. The optimization solver is chosen as NPSOL (see Gill, Murray, Saunders and Wright (1994)), and the program code is written in Fortran.

For the approximation step of the value function iteration method, we try the following four ways respectively: (1) the standard degree-19 Chebyshev polynomial interpolation method with 20 Chebyshev nodes; (2) order-4 B-splines method with 60 equally spaced nodes in $[0.1, 10]$; (3) the shape-preserving degree-19 Chebyshev polynomial approximation method with positive gradient and negative Hessian constraints, while the number of Chebyshev nodes is 20; (4) the

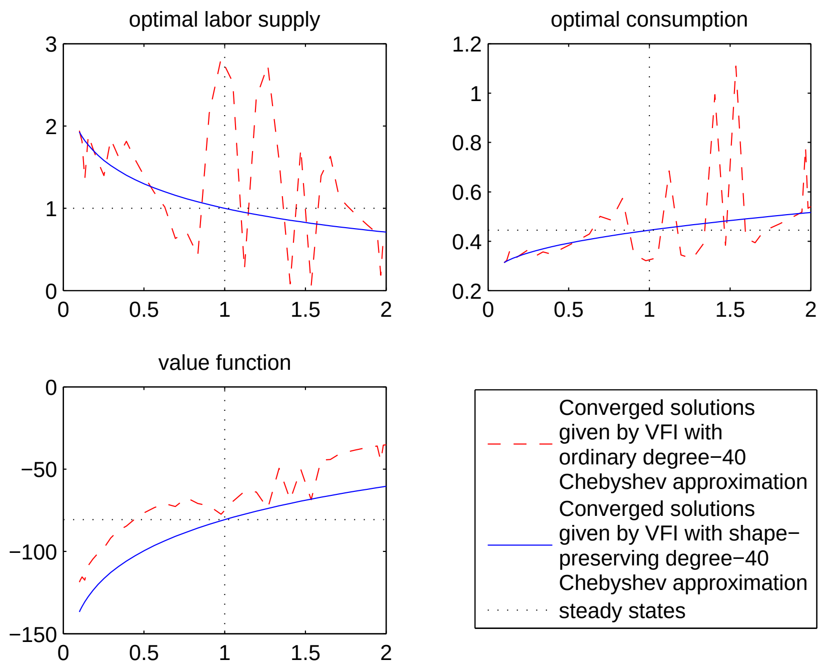


Figure 1: Example 1 of Numerical DP w/o shape-preserving approximation

revised Schumaker shape-preserving interpolation with 60 equally spaced nodes in $[0.1, 10]$;

Our results show that the value function iteration with either of the first two approximation methods diverges. However, the value function iteration with either of the last two shape-preserving approximation converges at the 133-rd step, their approximated optimal control functions and value functions are shown in Figure 2. We see that two converged value functions and the corresponding optimal control functions are very close to each other, and all the functions are smooth, monotone and concave. Moreover, when the capital state is $k^* = 1$, from the VFI with the shape-preserving Chebyshev polynomial approximation, the corresponding computed optimal $\hat{l}_C^* = 1.00388$, $\hat{c}_C^* = 0.44414$ and $\hat{V}(1)_C = -694.7552$. From the VFI with revised Schumaker shape-preserving interpolation, the corresponding computed optimal $\hat{l}_S^* = 0.98826$, $\hat{c}_S^* = 0.44538$ and $\hat{V}(1)_C = -702.7005$. All of these are very close to the exact optimal labor, consumption and value at the steady state.

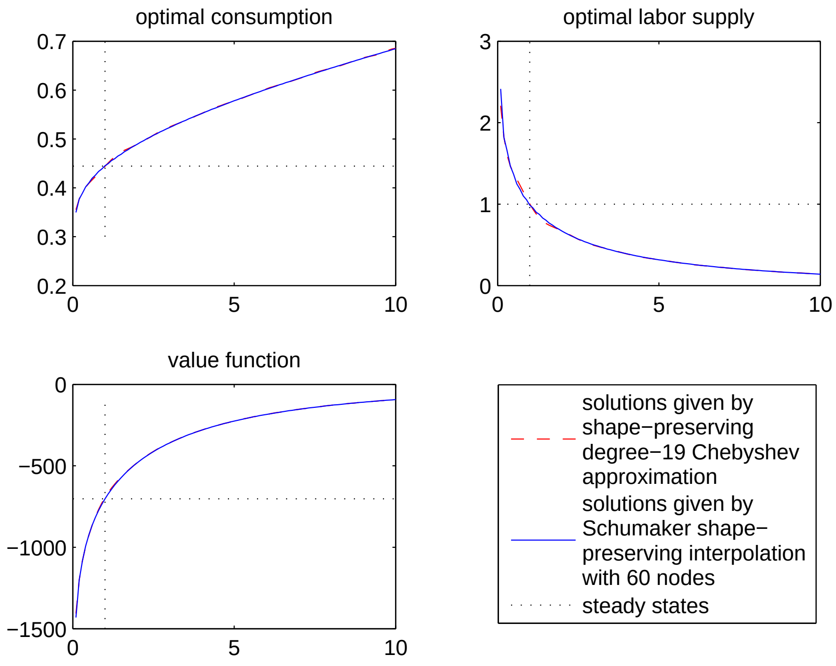


Figure 2: Example 2 of Numerical DP with shape-preserving approximation

6 Example for Multi-stage Portfolio Optimization Problems

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

$$W_{t+1} = R_f(W_t - e^\top S_t) + R^\top S_t, \tag{7}$$

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.,

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

where W_T is the terminal wealth derived from the recursive formula (7) 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

$$V_t(W) = \max_{S} \quad E\{V_{t+1}(R_f(W - e^\top S) + R^\top S)\},$$

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

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

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

where

$$P_{t+1,k} = P_{t,(k-1)/m+1} p_{(k \bmod m)+1}, \quad P_{0,j} = 1,$$

and

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

for $1 \leq k \leq m^{t+1}$ and $0 \leq t < T$.

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.

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 riskfree 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

$$u(W) = -(W - 0.2)^{-1}$$

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.

Since this example 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 3 shows the optimal bond allocation B_t and stock allocation S_t, for $t = 0, 1, \ldots, 5$. Note that the ranges of wealth are expanding over t. 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:

$$\begin{aligned} \underline{W}_{t+1} &= \min(R)\underline{W}_t = 0.9\underline{W}_t, \\ \overline{W}_{t+1} &= \max(R)\bar{W}_t = 1.4\overline{W}_t, \end{aligned}$$

where $\underline{W}_0 = 0.9$ and $\overline{W}_0 = 1.1$.

After obtaining these exact optimal allocations, we use them to test our algorithms' accuracy. The computational results of numerical DP algorithms are given by our AMPL code using MINOS optimization package (Murtagh and Saunders (1978)). And the ranges $[\underline{W}_t, \overline{W}_t]$ are given in the previous iterative way. Figure 4 shows the accuracy of rational function spline interpolation.

7 Conclusion

This paper presents a general shape-preserving DP algorithm and shows that shape-preserving property of the value function approximation is critical in convergence of the iteration methods and obtaining good solutions from numerical dynamic programming.

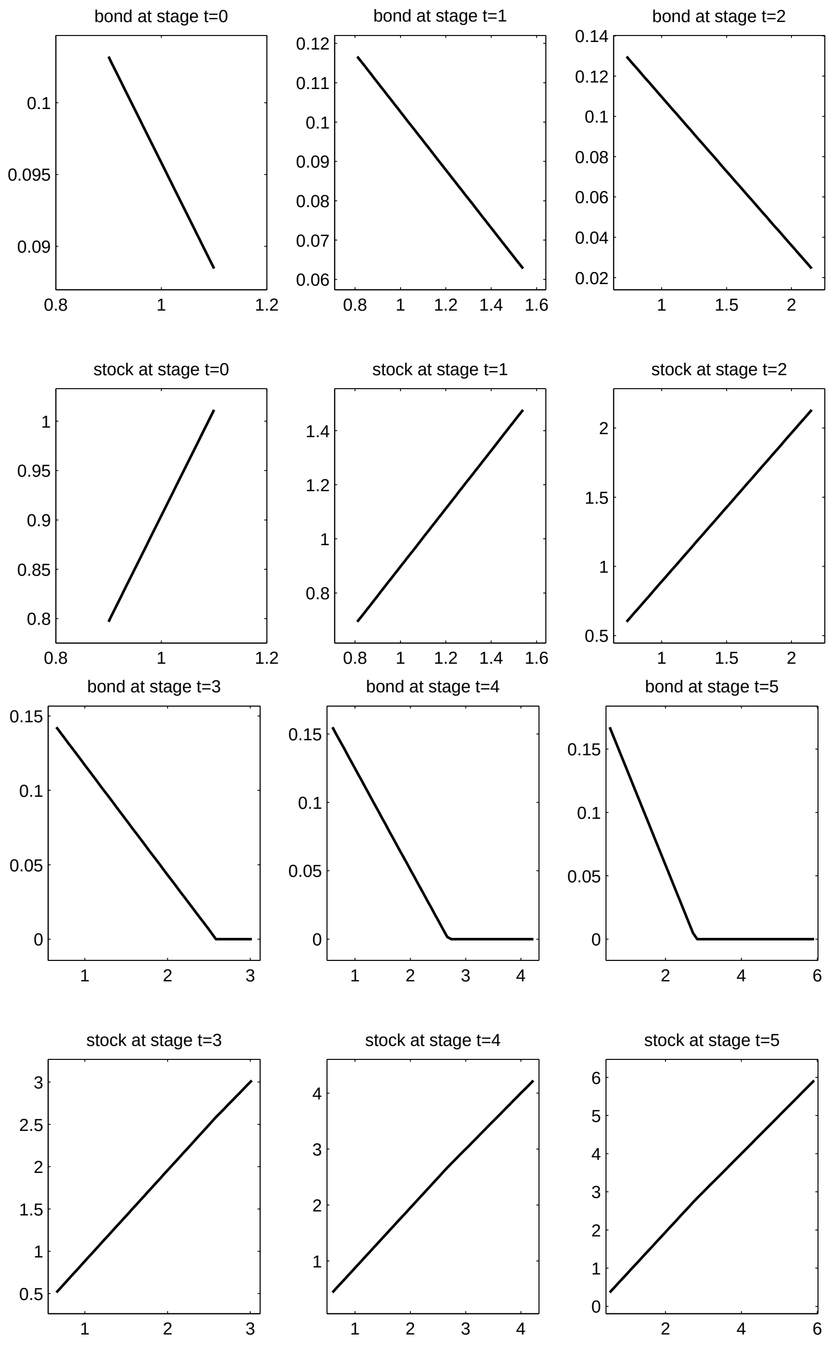
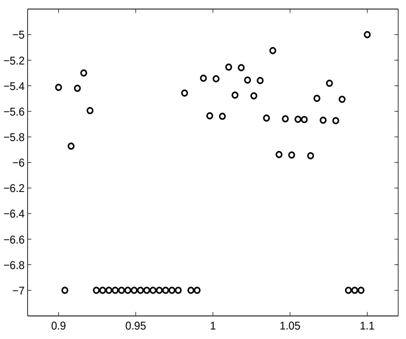
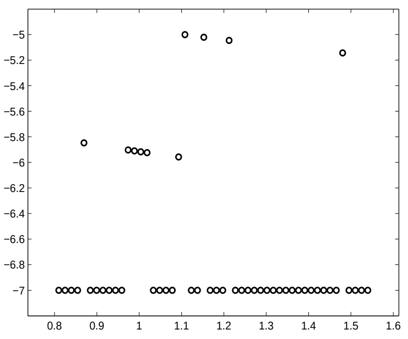


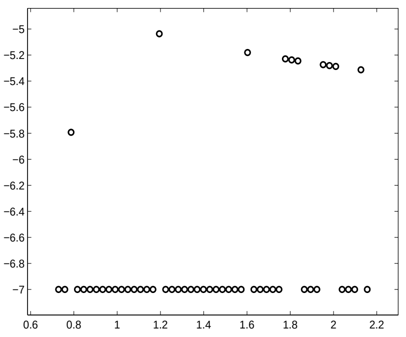
Figure 3: Exact optimal allocation and value functions



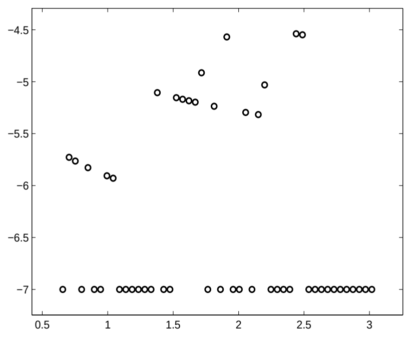
$\log_{10}$ (errors) of S_0



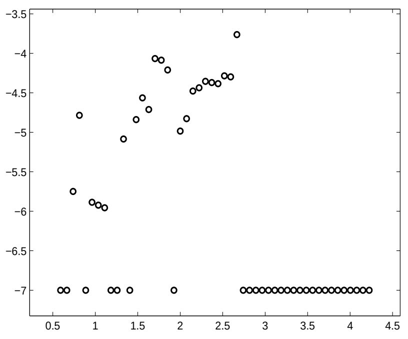
$\log_{10}$ (errors) of S_1



$\log_{10}$ (errors) of S_2



$\log_{10}$ (errors) of S_3



$\log_{10}$ (errors) of S_4

Figure 4: Relative Errors of Optimal Stock Allocations from Numerical DP with Rational Function Spline Interpolation

References

[1] Bellman, Richard (1957). *Dynamic Programming.* Princeton University Press.

[2] Byrd, Richard H., and Jorge Nocedal and Richard A. Waltz (2006). "KNITRO: an integrated package for nonlinear optimization".

[3] Cai, Yongyang (2009). *Dynamic Programming and Its Application in Economics and Finance.* PhD thesis, Stanford University.

[4] 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.

[5] Cai, Yongyang, and Kenneth Judd (2011). "Dynamic programming with Hermite information".

[6] Fiorot, J.C., and J. Tabka (1991). "Shape-preserving C^2 cubic polynomial interpolating splines". *Mathematics of Computation,* 57(195), 291–298.

[7] Gill, Philip, Walter Murray, Michael Saunders, and Margaret Wright (1994). "Users Guide for NPSOL 5.0: a Fortran Package for Nonlinear Programming". Technical report, SOL, Stanford University.

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

[9] Judd, Kenneth (1998). *Numerical Methods in Economics.* The MIT Press.

[10] Judd, Kenneth, and Andrew Solnick (1994). "Numerical dynamic programming with shape-preserving splines".

[11] Maldonado, Wilfredo L., and Benar Fux Svaiter (2001). "On the accuracy of the estimated policy function using the Bellman contraction method". *Economics Bulletin*, 3(15), 1–8.

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

[13] Pruess, Steven (1993). "Shape-preserving C^2 cubic spline interpolation". *IMA Journal of Numerical Analysis*, 13, 493–507.

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

[15] Santos, Manuel S., and Jesus Vigo-Aguiar (1998). "Analysis of a numerical dynamic programming algorithm applied to economic models". *Econometrica*, 66(2), 409426.

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

[17] Wang, Sheng-Pen, and Kenneth L. Judd (2000). "Solving a savings allocation problem by numerical dynamic programming with shape-preserving interpolation". *Computers & Operations Research*, 27(5), 399408.