Dynamic Programming with Piecewise Linear Interpolation

Kenneth Judd and Yongyang Cai

April 8, 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 [3], and Maldonado [2].

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 [1]).

A general DP model is based on the Bellman equation:

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

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

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

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.

2.1 Piecewise Linear Interpolation

If Lagrange data $\{(x_i, v_i) : i = 1, \ldots, m\}$ is given, then its piecewise linear interpolation is

$$\hat{V}(x) = b_{j,0} + b_{j,1}x, \quad \text{if } x \in [x_j, x_{j+1}],$$

where

$$\begin{aligned} b_{j,1} &= \frac{v_{j+1} - v_j}{x_{j+1} - x_j}, \\ b_{j,0} &= v_i - b_{j,1}x_i, \end{aligned}$$

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

In the maximization step of numerical DP algorithms, a naive way is to let optimization software to directly solve the maximization problem

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

where x_i^+ is the next-stage state (may be random) conditional on the current-stage state x and the action a. Let the transition function from state x to x^+ be given as $x^+ = g(x, a, z)$, where z is the random part to make x^+ random (e.g., the random return R in the portfolio optimization problem).

When z is a discrete random variable, we assume that $z = z_1, \ldots, z_n$ with probabilities $p_1, \ldots, p_n$. When z is a continuous random variable and a numerical quadrature formula is chosen to compute $E\{\hat{V}(x_i^+; \mathbf{b}^{t+1}) \mid x_i, a_i\}$, we assume that its discretization is $z = z_1, \ldots, z_n$ with weights $p_1, \ldots, p_n$. See Judd (1998) and Cai (2009) for detailed discussion of numerical quadrature formulas.

Let $\{x_j^{t+1} : \ j = 1, \ldots, m\}$ be the pre-specified nodes at stage $t+1$. Let parameters $\mathbf{b}^{t+1}$ of piecewise linear interpolation be $b_{j,0}^{(t+1)}, b_{j,1}^{(t+1)}$ on $[x_j^{t+1}, x_{j+1}^{t+1}]$ for $j = 1, \ldots, m-1$. So the above model becomes

$$\begin{aligned} v_i^t \quad = \quad & \max_{a_i \in \mathcal{D}(x_i,t), x_{ik}^+} \quad u_t(x_i, a_i) + \beta \sum_{k=1}^{n} p_k \hat{V}(x_{ik}^+; \mathbf{b}^{t+1}), \\ & \text{s.t.} \qquad x_{ik}^+ = g(x_i, a, z_k), \quad 1 \le k \le n. \end{aligned} \tag{1}$$

2.2 Min-Function Approach

Since this piecewise linear interpolation $\hat{V}(x; \mathbf{b}^{t+1})$ is not differentiable at the nodes $x_j^{(t+1)}$, it will be difficult for most optimization methods to find optimal solutions of the above model.

This challenge could be solved by modifying the problem (1) as follows:

$$\begin{aligned} v_i^t \quad = \quad & \max_{a_i \in \mathcal{D}(x_i,t), y_{ik}, x_{ik}^+} \; u_t(x_i, a_i) + \beta \sum_{k=1}^{n} p_k y_{ik}, \\ & \text{s.t.} \qquad x_{ik}^+ = g(x_i, a, z_k), \quad 1 \le k \le n, \\ & \qquad\quad y_{ik} \le b_{j,0}^{(t+1)} + b_{j,1}^{(t+1)} x_{ik}^+, \quad 1 \le j < m, \;\; 1 \le k \le n. \end{aligned} \tag{2}$$

The objective function is smooth and inequality constraints are linear and sparse so that we can apply fast Newton type optimization algorithms to solve this problem if g is also smooth. Although this new model adds $(m-1)n$ linear inequality constraints, only several of them will be active, such that optimization softwares can still solve the new model quickly.

Moreover, this way does not need to find the interval where x_{ik}^+ locates, while the naive way has to. It has more advantages for multi-dimensional approximation, because it is harder to find the polytope where x_{ik}^+ locates.

2.3 Convex-Set Approach

Both model (1) and (2) need to calculate coefficients explicitly, and this is complicated for multi-dimensional piecewise linear interpolation.

The following model has no need to compute the coefficients explicitly:

$$\begin{aligned} v_i^t \quad = \quad & \max_{a_i \in \mathcal{D}(x_i,t), \mu_{jk} \ge 0, y_{ik}, x_{ik}^+} \; u_t(x_i, a_i) + \beta \sum_{k=1}^{n} p_k y_{ik}, \\ & \text{s.t.} \qquad x_{ik}^+ = g(x_i, a, z_k), \qquad 1 \le k \le n, \\ & \qquad\quad x_{ik}^+ \le \sum_{j=1}^{m} \mu_{jk} x_j^{(t+1)}, \quad 1 \le k \le n, \\ & \qquad\quad y_{ik} = \sum_{j=1}^{m} \mu_{jk} v_j^{(t+1)}, \quad 1 \le k \le n. \\ & \qquad\quad \sum_{j=1}^{m} \mu_{jk} = 1, \quad 1 \le k \le n. \end{aligned} \tag{3}$$

In comparison with the model (2), this model has mn more control variables (μ_{jk}), but has many fewer constraints (not counting box constraints for control variables).

Since this model does not need values of $\mathbf{b}^{(t+1)}$, it saves the hard fitting step for multi-dimensional piecewise linear interpolation. Moreover, in multi-dimensional cases this model will preserve concavity of value functions, but the other two models can only at great cost. The disadvantage of this model is that it may be slower.

3 Example for Multi-stage Portfolio Optimization Problems

We 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

$$W_{t+1} = R_f(W_t - e^\top S_t) + R^\top S_t, \tag{4}$$

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 (4) 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

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

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 $1 \le j \le m$, 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 $1 \le k \le m^{t+1}$ and $0 \le t < T$.

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., $B_t \ge 0$ and $S_t \ge 0$ 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 1 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 $\overline{W}_0 = 1.1$.

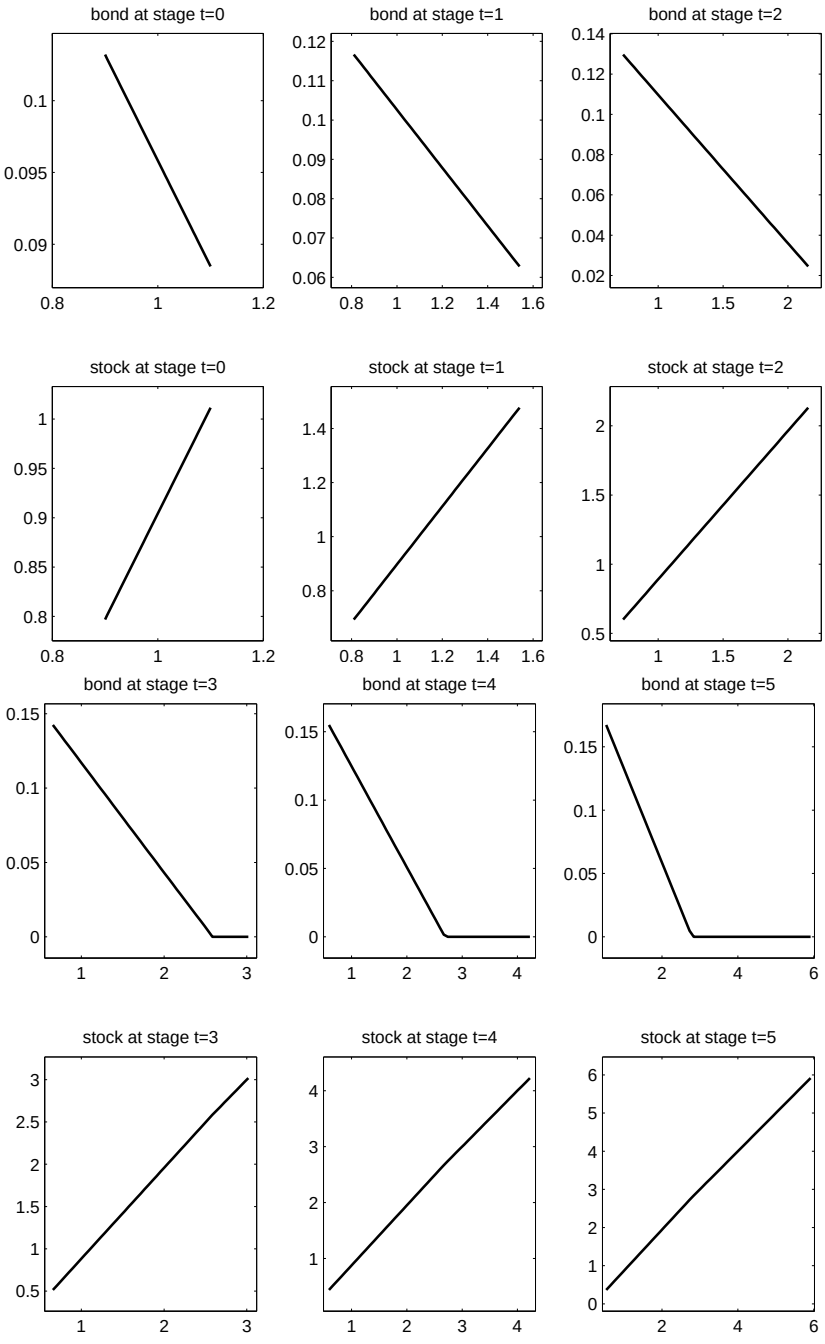

Figure 1: Exact optimal allocation and value functions

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 Fortran code using NPSOL (see Gill, Murray, Saunders and Wright (1994)). And the ranges $[\underline{W}_t, \overline{W}_t]$ are given in the previous iterative way.

We ran numerical DP algorithm with piecewise linear interpolation on 100 equally spaced nodes using model (1), (2) and (3) respectively. Figure 2 shows relative errors of the computed optimal stock allocations by the algorithms in comparison with the exact solutions. We omit the figure of relative errors of model (3) using the convex-set approach, because its errors are almost the same with the model (2) using the min-function approach. We see that the min-function approach really helps to improve the accuracy of the solutions.

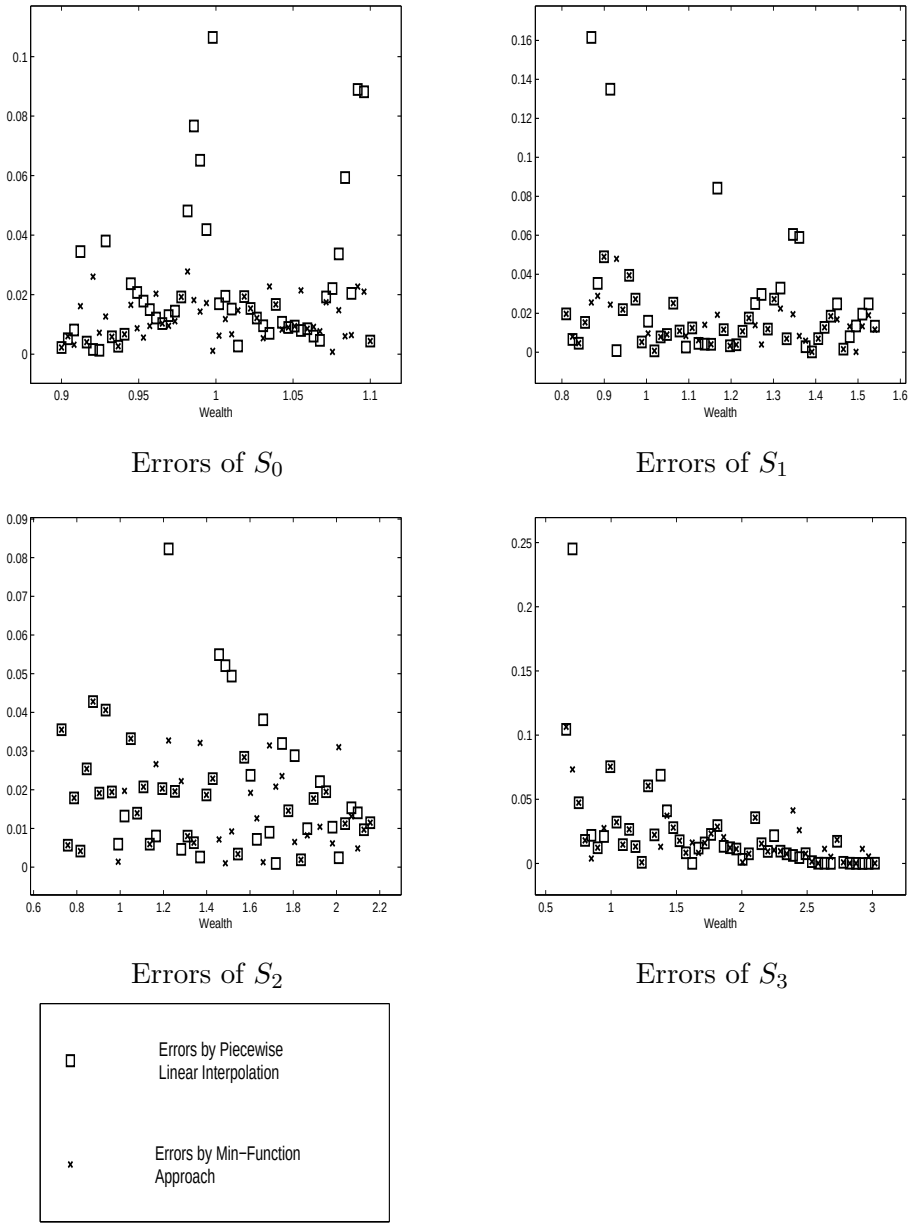

Figure 2: Relative Errors of Optimal Stock Allocations from Numerical DP with Piecewise Linear Interpolation vs Min-Function Approach

References

[1] Kenneth Judd. *Numerical Methods in Economics*. The MIT Press, 1998.

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

[3] Manuel S. Santos and Jesus Vigo-Aguiar. Analysis of a numerical dynamic programming algorithm applied to economic models. *Econometrica*, 66(2):409–426, 1998.