Math Meth Oper Res (2015) 81:245–267
DOI 10.1007/s00186-015-0495-z

ORIGINAL ARTICLE

Dynamic programming with Hermite approximation

Yongyang Cai · Kenneth L. Judd

Received: 22 January 2014 / Accepted: 5 February 2015 / Published online: 13 February 2015

Abstract Numerical dynamic programming algorithms typically use Lagrange data to approximate value functions over continuous states. Hermite data can be easily obtained from solving the Bellman equation and used to approximate the value functions. We illustrate the use of Hermite data with one-, three-, and six-dimensional examples. We find that Hermite approximation improves the accuracy in value function iteration (VFI) by one to three digits using little extra computing time. Moreover, VFI with Hermite approximation is significantly faster than VFI with Lagrange approximation for the same accuracy, and this advantage increases with the dimension of the continuous states.

Keywords Dynamic programming · Value function iteration · Hermite approximation · Dynamic portfolio optimization · Multi-country optimal growth model

JEL Classification C61 · C63

We are grateful to the editor, an associate editor, and two anonymous referees for their helpful comments. We also thank Jesus Fernandez-Villaverde, Thomas Lontzek, and Jevgenijs Steinbuks for their comments. Cai gratefully acknowledges NSF support (SES-0951576). An earlier version of this paper is "Dynamic Programming with Hermite Interpolation" (RDCEP Working Paper 12-09).

Y. Cai (✉) · K. L. Judd
Hoover Institution, Stanford University, 424 Galvez Mall, Stanford, CA 94305, USA
e-mail: yycai@stanford.edu

Y. Cai
University of Chicago, Chicago, IL, USA

K. L. Judd
NBER, Cambridge, MA, USA

1 Introduction

Dynamic optimization problems in economics are generally formulated in terms of dynamic programming (DP).[1] Value function iteration (VFI) is often used to solve DP problems, but it is often numerically challenging due to nonlinearities and/or dimensionality of the DP problems. VFI proceeds in a backwards recursion manner, beginning with the specified value function at the terminal time T. At each time $t < T$, VFI first solves the Bellman equation at a finite number of approximation nodes, and then uses the optimal values, also known as the Lagrange data, to construct an approximation of the time t value function; we call that approach L-VFI for Lagrange Value Function Iteration.

This paper uses the Envelope Theorem to reformulate the Bellman optimization problem to get both the levels and gradients of the value function at the approximation nodes, which together comprise the Hermite data, with no increase in computational effort. We then use a Hermite approximation method to construct an approximation to the value function; we call this approach H-VFI for Hermite value function iteration. We demonstrate that H-VFI is significantly more efficient than L-VFI for some common economics problems. The efficiency of H-VFI is particularly important for finite-horizon problems, such as life-cycle consumption problems (e.g., Cocco et al. 2005), that can be solved only by some form of VFI. Our results are consistent with the simple intuition that knowing the value and the gradient of a d-dimensional function at a point is roughly equivalent to knowing the values of the function at $d + 1$ points.

In this paper, we show the efficiency of H-VFI with two kinds of examples: dynamic portfolio problems and multi-country optimal growth problems.[2] For these examples we approximate the value functions with Chebyshev polynomials. The approximation scheme consists of two parts: basis functions and approximation nodes. The approximation nodes can be chosen as uniformly spaced nodes, Chebyshev nodes, or some other specified nodes. Approximation methods can be classified as either spectral methods or finite element methods. A spectral method uses globally nonzero basis functions $\phi_j(\mathbf{x})$, such that $\hat{V}(\mathbf{x}; \mathbf{b}) = \sum_{j=0}^{n} b_j \phi_j(\mathbf{x})$ is a degree n approximation. Examples of spectral methods include ordinary polynomial approximation, Chebyshev polynomial approximation, and shape-preserving Chebyshev polynomial approximation (Cai and Judd 2013). In contrast, a finite element method uses locally basis functions $\phi_j(\mathbf{x})$ that are nonzero over sub-domains of the approximation domain. Examples of finite element methods include piecewise linear interpolation, cubic splines, and B-splines. See Rivlin (1990), Judd (1998), and Cai et al. (2010) for more details. In Appendices A

[1] Examples include optimal growth problems (e.g., Trick and Zin 1997; Judd 1998; Den Haan et al. 2011), life-cycle consumption problems (e.g., Cocco et al. 2005), dynamic portfolio problems (e.g., Gupta and Murray 2005; Infanger 2006; Cai et al. 2013), dynamic resource allocation problems (e.g., Powell and Van Roy 2004), and optimal carbon tax problems (e.g., Cai et al. 2013a). See Judd (1998), Bertsekas (2005, 2007), Powell (2007), Rust (2008), and Cai and Judd (2014) for more examples.

[2] H-VFI can also be more efficient than L-VFI in solving other kinds of DP problems with differential value functions over continuous state variables, such as life-cycle consumption problems (e.g., Cocco et al. 2005), dynamic resource allocation problems (e.g., Powell and Van Roy 2004), and optimal carbon tax problems (e.g., Cai et al. 2013a).

and B we describe one-dimensional Chebyshev polynomials and multi-dimensional complete Chebyshev polynomials, respectively. For one-dimensional problems, the Hermite approximation can also be used in combination with spline methods, such as the Schumaker shape-preserving spline method (Schumaker 1983) or the rational function spline method (Cai and Judd 2012). For multi-dimensional problems, we use complete Chebyshev polynomials to fit our Hermite data, but we could also use many alternatives such as multivariate splines and radial basis functions, as H-VFI does not depend on a specific functional form used for approximating the value functions.

The key idea in H-VFI is generating and using gradient information. Some of the ideas presented in this paper have been discussed in the literature, but as far as the authors know, this is the first complete discussion of the issues and analysis of its performance on a range of common problems in economics. Philbrick and Kitanidis (2001) describe the generation and the use of Hermite information but do not invoke the Envelope Theorem to construct the gradients. Instead, they propose computing the gradients in a much more costly manner. They also use tensor products of splines for approximation, an approach that suffers from a curse of dimensionality. We propose a complete polynomial approximation method, for which this "curse-of-dimensionality" problem does not exist.[3] Philbrick and Kitanidis (2001) report at most three-digit accuracy for their examples of dimension four or less, but less than two digits for higher dimensions. Our examples achieve higher accuracy. Wang and Judd (2000) combine Hermite approximation with shape preservation for two-dimensional problems, but use a much less efficient approach to approximation. The rational function spline method (Cai and Judd 2012) also uses Hermite interpolation, but that method only works for one-dimensional problems, and produces only C^1 approximations of the value functions. Moreover, none of these papers document the actual performance of H-VFI relative to L-VFI for economics problems.

The paper is organized as follows. Section 2 outlines the basic numerical L-VFI algorithm with Chebyshev polynomials. Section 3 presents the H-VFI algorithm. Section 4 gives some numerical examples, where we compare the efficiency of Lagrange and Hermite value function iteration. Section 5 concludes.

2 Review of numerical methods for DP

In a general DP problem, we let $\mathbf{x}$ denote the vector of continuous states, and let θ denote the vector of discrete states. We use $\mathbf{x}^+$ and θ^+ to denote the vectors of continuous and discrete states in the next period. Numerical solutions to a DP problem are based on the Bellman equation (Bellman 1957):

[3] Rust (1997) and Rust et al. (2002) prove that the curse of dimensionality exists for dynamic programming problems in the worst-case analysis. However, Griebel and Wozniakowski (2006) show that there is no curse of dimensionality in approximating functions having sufficient smoothness, while many DP problems in economics have smooth value functions which can be approximated well by the complete polynomial approximation method.

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

for $t = 0, 1, ..., T-1$, where $\mathbf{a}$ is the vector of action variables constrained by $\mathbf{a} \in \mathcal{D}(\mathbf{x},\theta,t)$, ω and ϵ are vectors of random variables, and g_t and h_t are laws of motion for states $\mathbf{x}$ and θ at time t. We call $V_t(\mathbf{x},\theta)$ the value function at time t. The terminal value function $V_T(\mathbf{x},\theta)$ is given. Here, $u_t(\mathbf{x},\mathbf{a})$ is the utility function at time t, β is the discount factor, and $\mathbb{E}_t\{\cdot\}$ is the expectation operator conditional on time-t information.

In the simpler case where there are only continuous states with deterministic transition laws, the Bellman equation (1) becomes

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

Without loss of generality, we will use the simpler DP case to describe our algorithms.

If state and control variables in a DP problem are continuous, then the value functions must be approximated in some computationally tractable manner. It is common to approximate the value functions with a finitely parameterized collection of functions; that is, we use some functional form $\hat{V}(\mathbf{x};\mathbf{b})$, where $\mathbf{b}$ is a vector of parameters, and approximate the value function, $V(\mathbf{x})$, with $\hat{V}(\mathbf{x};\mathbf{b})$ for some parameter value $\mathbf{b}$. For example, $\hat{V}$ could be a linear combination of polynomials, where $\mathbf{b}$ would be the weights on polynomials. After the functional form is specified, we focus on finding the vector of parameters, $\mathbf{b}$, such that $\hat{V}(\mathbf{x};\mathbf{b})$ approximately satisfies the Bellman equation (2). The following algorithm gives the traditional value function iteration to solve the deterministic DP problem (2).

Algorithm 1. *Lagrange Value Function Iteration* (L-VFI)

Initialization. *Choose the approximation nodes, $\mathbb{X}_t = \{\mathbf{x}_{t,i} : 1 \le i \le N_t\} \subset \mathbb{R}^d$, for every $t < T$, and choose a functional form for $\hat{V}(\mathbf{x};\mathbf{b})$. Let $\hat{V}(\mathbf{x};\mathbf{b}_T) \equiv V_T(\mathbf{x})$. Then for $t = T-1, T-2, \ldots, 0$, iterate through steps 1 and 2.*

Step *1. Maximization Step. Compute*

$$v_i = \max_{\mathbf{a}\in\mathcal{D}(\mathbf{x}_i,t)} u_t(\mathbf{x}_i,\mathbf{a}) + \beta\hat{V}(\mathbf{x}^+;\mathbf{b}_{t+1}) \tag{3}$$
$$\text{s.t.} \quad \mathbf{x}^+ = g_t(\mathbf{x}_i,\mathbf{a}),$$

for each $\mathbf{x}_i \in \mathbb{X}_t$, $1 \le i \le N_t$.

Step *2. Fitting Step. Using an appropriate approximation method, compute $\mathbf{b}_t$ such that $\hat{V}(\mathbf{x};\mathbf{b}_t)$ approximates $(\mathbf{x}_i, v_i)$ data.*

Algorithm 1 shows that there are two main components in value function iteration for the deterministic DP problems: optimization and approximation. In this paper we focus on approximation methods. Detailed discussion of numerical DP can be found in Cai et al. (2010), Judd (1998) and Rust (2008).

3 Hermite value function iteration algorithm

L-VFI is the traditional approach to value function iteration. In this section, we show how to generate more information in each maximization step that will allow us to construct better value function approximations. More specifically, we will first show that each maximization problem in the maximization step of L-VFI can produce the gradient of the value function at any approximation node. Then we show how to use the gradient information to produce better value function approximations.

3.1 H-VFI

Traditional approximation methods (L-VFI) in DP problems use only Lagrange data $\{(\mathbf{x}_i, v_i) : i = 1, \ldots, N\}$ for pre-specified approximation nodes $\{\mathbf{x}_i\}$. However, the optimization problem (3) in L-VFI can give us more information, not just v_i. We can also let the optimization problem (3) produce the gradient of the value function at $\mathbf{x}_i$ at almost no cost if the value function is differentiable. This is done by invoking the Envelope Theorem (see, e.g., Judd 1998), which we next state.

Theorem 1 (Envelope Theorem) *Let*

$$\begin{aligned} H(\mathbf{x}) = \max_{\mathbf{a}} \; & f(\mathbf{x}, \mathbf{a}) \\ & s.t.\ g(\mathbf{x}, \mathbf{a}) = 0, \\ & (\mathbf{x}, \mathbf{a}) \geq 0, \end{aligned} \tag{4}$$

where $\mathbf{x} = (x_1, ..., x_d) \in \mathbb{R}^d$. *Suppose that* $\mathbf{a}^*(\mathbf{x})$ *is the optimizer of (4). Let* $\lambda^*(\mathbf{x})$ *be the vector of shadow prices for the equality constraints* $g(\mathbf{x}, \mathbf{a}) = 0$, *and let* $\mu^*(\mathbf{x})$ *be the vector of shadow prices of the inequality constraints* $h(\mathbf{x}, \mathbf{a}) \geq 0$. *Then*

$$\frac{\partial H(\mathbf{x})}{\partial x_j} = \frac{\partial f}{\partial x_j}(\mathbf{x}, \mathbf{a}^*(\mathbf{x})) + \lambda^*(\mathbf{x})^\top \frac{\partial g}{\partial x_j}(\mathbf{x}, \mathbf{a}^*(\mathbf{x})) + \mu^*(\mathbf{x})^\top \frac{\partial h}{\partial x_j}(\mathbf{x}, \mathbf{a}^*(\mathbf{x})), \tag{5}$$

for $j = 1, \ldots, d$.

The Envelope Theorem can be applied to the Bellman equation (2) at an approximation node $\mathbf{x}_i$, and then we could use the formula (5) to generate Hermite data of the value function V_t at $\mathbf{x}_i$. But the formula (5) requires the computation of the gradients of f, g, and h. Corollary 1 shows how to reformulate the optimization problem so that an optimization solver can directly output the value of $\partial H(\mathbf{x})/\partial x_j$.

Corollary 1 *The optimization problem (4) is equivalent to*

$$\begin{aligned} H(\mathbf{x}) = \max_{\mathbf{a},\mathbf{y}} \; & f(\mathbf{y}, \mathbf{a}) \\ & s.t.\ g(\mathbf{y}, \mathbf{a}) = 0, \\ & h(\mathbf{y}, \mathbf{a}) \geq 0, \\ & x_j - y_j = 0, \quad j = 1, \ldots, d. \end{aligned} \tag{6}$$

Therefore,

$$\frac{\partial H(\mathbf{x})}{\partial x_j} = \tau_j^*(\mathbf{x}),$$

where $\tau_j^*(\mathbf{x})$ *is the shadow price of the trivial constraint* $x_j - y_j = 0$*, for* $j = 1, \ldots, d$.

Corollary 1 takes the optimization problem (4), adds variables y_j and constraints $x_j - y_j = 0$, and replaces the vector $\mathbf{x}$ by the vector $\mathbf{y}$ in the objective function and all constraints. The Envelope Theorem tells us that $\partial H(\mathbf{x})/\partial x_j$ is the shadow price of the trivial constraint $x_j - y_j = 0$. Any modern solver will include this shadow price in the output if the user requests that information.[4] The extra cost of the reformulated problem is trivial for any modern solver. This procedure eliminates the need to compute the costly formula (5).

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

Algorithm 2. *Hermite value function iteration* (H-VFI)

Initialization. *Choose the approximation nodes,* $\mathbb{X}_t = \{\mathbf{x}_{t,i} : 1 \le i \le N_t\} \subset \mathbb{R}^d$*, for every* $t < T$*, and choose a functional form for* $\hat{V}(\mathbf{x}; \mathbf{b})$*. Let* $\hat{V}(\mathbf{x}; \mathbf{b}_T) \equiv V_T(\mathbf{x})$*. Then for* $t = T-1, T-2, \ldots, 0$*, iterate through steps 1 and 2.*

Step *1. Maximization Step. For each* $\mathbf{x}_i = \left(x_{i,1}, ..., x_{i,d}\right) \in \mathbb{X}_t$, $1 \le i \le N_t$*, compute*

$$\begin{aligned} v_i = \max_{\mathbf{a} \in \mathcal{D}(\mathbf{y},t), \mathbf{y}} \quad & u_t(\mathbf{y}, \mathbf{a}) + \beta \hat{V}(\mathbf{x}^+; \mathbf{b}_{t+1}), \\ \text{s.t.} \quad & \mathbf{x}^+ = g_t(\mathbf{y}, \mathbf{a}), \\ & x_{i,j} - y_j = 0, , \quad j = 1, \ldots, d, \end{aligned}$$

and

$$\mathbf{s}_i = \left(\tau_1^*(\mathbf{x}_i), ..., \tau_d^*(\mathbf{x}_i)\right),$$

where $\tau_j^*(\mathbf{x}_i)$ *is the shadow price of the constraint* $x_{i,j} - y_j = 0$.[5]

Step *2. Hermite Fitting Step. Using an appropriate approximation method, compute* $\mathbf{b}_t$ *such that* $\hat{V}(\mathbf{x}; \mathbf{b}_t)$ *approximates* $(\mathbf{x}_i, v_i, \mathbf{s}_i)$ *data.*

[4] It is standard for a solver to report the multipliers associated with a solution of a constrained optimization problem. For example, in fmincon, the solver in the Matlab Optimization Toolbox, the command

[X,FVAL,EXITFLAG,OUTPUT,LAMBDA] = fmincon(FUN, X0, ...)

prompts the solver to report the multipliers in the vector LAMBDA. Similar commands are provides by other solvers, such as KNITRO (Byrd et al. 2006), CONOPT (Drud 1996), SNOPT (Gill et al. 2005), MINOS (Murtagh and Saunders 2003), and so on. Moreover, these solvers are available in the NEOS server (Czyzyk et al. 1998; Gropp and Moré 1997) by uploading code in AMPL (Fourer et al. 1990, 2003) or GAMS (Bisschop and Meeraus 1982; Brooke et al. 1997; McCarl 2011) with the NEOS graphical user interface (http://www.neos-server.org/neos/) or a callable interface (Dolan et al. 2008).

[5] For problems with a simple formulation of (5), such as the dynamic portfolio problem (9), we will use the formula (5) to generate the Hermite data.

We can easily extend the above algorithm to solve the general stochastic DP model (1). Moreover, this H-VFI method can also be efficiently parallelized to solve high-dimensional problems like what the L-VFI method did in Cai et al. (2015).

3.2 Hermite approximation methods

Many approximation methods can use Hermite information. Our examples have smooth value functions, as is often the case for economics problems. In this section, we describe polynomial approximation methods that produce smooth approximations using the Hermite information. We next introduce Chebyshev–Hermite interpolation for one dimensional cases, and Hermite approximation with complete Chebyshev polynomials for multi-dimensional problems.

3.2.1 One-dimensional Chebyshev–Hermite interpolation

Chebyshev interpolation is often used in L-VFI, and can also be used for H-VFI. For the one-dimensional approximation in this paper, we use the interpolation only, so we just discuss the one-dimensional interpolation using Hermite information in the following.

Assume that we have Hermite data $\{(x_i, v_i, s_i) : i = 1, \ldots, m\}$ on $[x_{\min}, x_{\max}]$, where $x_i = (z_i + 1)(x_{\max} - x_{\min})/2 + x_{\min}$ (with $z_i = -\cos((2i-1)\pi/(2m))$) are the Chebyshev nodes, $v_i = V(x_i)$ and $s_i = V'(x_i)$. The following system of $2m$ linear equations can produce coefficients for the degree $2m-1$ Chebyshev polynomial interpolation (see "Appendix 1") on the Hermite data:

$$\begin{cases} \sum_{j=0}^{2m-1} b_j \mathcal{T}_j(z_i) = v_i, & i = 1, \ldots, m, \\ \frac{2}{x_{\max} - x_{\min}} \sum_{j=1}^{2m-1} b_j \mathcal{T}_j'(z_i) = s_i, & i = 1, \ldots, m, \end{cases} \tag{7}$$

where $\mathcal{T}_j(z)$ are Chebyshev basis polynomials (see "Appendix 1"). After the coefficients are computed from solving the linear system (7), we can use the degree $2m-1$ Chebyshev polynomial (17) to approximate $V(x)$.

3.2.2 Multidimensional Hermite approximation with complete Chebyshev polynomials

We can generalize the idea of combining Chebyshev polynomials and Hermite data to higher dimensions. We use a least squares approach to fit complete Chebyshev polynomials to Hermite data.

For simplicity, we assume that the approximation domain is $[-1, 1]^d$ in a d-dimensional approximation problem. It can be easily extended to any domain $[\mathbf{x}_{\min}, \mathbf{x}_{\max}] \subset \mathbb{R}^d$. Let us assume that we have Hermite data $\{(\mathbf{x}_i, v_i, \mathbf{s}_i) : i = 1, \ldots, N\}$ on $[-1, 1]^d$, where $\mathbf{x}_i \in [-1, 1]^d$ is the i-th approximation node, $v_i = V(\mathbf{x}_i)$, and $\mathbf{s}_i = (s_{i,1}, \ldots, s_{i,d})$ is the gradient of V at $\mathbf{x}_i$, i.e., $s_{i,j} = \frac{\partial}{\partial x_j} V(\mathbf{x}_i)$. The

following least squares model can produce coefficients for the degree n complete Chebyshev polynomial approximation (see "Appendix 2") on the Hermite data:

$$\min_{\mathbf{b}} \left\{ \sum_{i=1}^{N} \left(v_i - \sum_{0 \le |\alpha| \le n} b_\alpha \mathcal{T}_\alpha (\mathbf{x}_i) \right)^2 + \sum_{i=1}^{N} \sum_{j=1}^{d} \left(s_{i,j} - \sum_{0 \le |\alpha| \le n} b_\alpha \frac{\partial}{\partial x_j} \mathcal{T}_\alpha (\mathbf{x}_i) \right)^2 \right\}. \tag{8}$$

It is easy to solve the least squares problem (8) to get the coefficients $\mathbf{b}$ of the Hermite approximation as it has no constraints.[6] When the approximation nodes are tensor grids with m nodes in each dimension, there are $N = m^d$ approximation nodes $\{\mathbf{x}_i\}$, so there are $(d+1)m^d$ information data $\{v_i, \mathbf{s}_i\}$ used for computing the coefficients of the degree n complete Chebyshev polynomial approximation. In this case, it is generally better to use degree $n = 2m - 1$ complete Chebyshev polynomials for the Hermite approximation, as the number of the unknown coefficients is $\binom{2m-1+d}{d}$, which is less than the number of the information data, $(d+1)m^d$, for any $d \ge 2$ and $m > 1$.[7] In our examples, when we apply H-VFI, we always use Hermite data on $N = m^d$ approximation nodes and degree $n = 2m - 1$ complete Chebyshev polynomials, and then solve the least squares problem (8) to compute the coefficients of our Hermite approximation.

4 Applications

This section applies the H-VFI algorithm to solve multi-stage portfolio optimization problems using one-dimensional Hermite interpolation, and to solve multi-dimensional optimal growth problems using Hermite approximation with complete Chebyshev polynomials.

4.1 Multi-stage portfolio optimization

We next compare H-VFI and L-VFI by solving multi-stage portfolio optimization problems. The dynamic portfolio optimization problem assumes that there are D stocks and one bond available for investment over the stages $t = 0, 1, \ldots, T-1$. The investor's objective is to maximize the expected utility of wealth at the terminal stage T. For each period, the bond has a risk-free return $R_f = e^r$ where r is the interest rate, and the stocks have a multivariate random return vector $\mathbf{R} = (R_1, \ldots, R_D)^\top$. This is a standard financial problem.

In each period t, the investor's portfolio has market value wealth W_t; we assume no transaction costs, implying that W_t can be used as the unique state variable. In period t, the investor chooses a new portfolio $\mathbf{S}_t = \left(S_{t,1}, \ldots, S_{t,D}\right)^\top$ where $S_{t,i}$ is the market value of the i-th stock. The amount of wealth invested in the bond is $B_t = W_t - \mathbf{e}^\top \mathbf{S}_t$

[6] The objective in (8) is an unweighted sum of squared errors. Weighted versions of the least squares may do better but we do not pursue that possibility in this paper.

[7] This can be proved recursively.

where $\mathbf{e}$ is the D-dimensional column vector of ones. Thus, the wealth at time $t+1$ is

$$W_{t+1} = R_f B_t + \mathbf{R}^\top \mathbf{S}_t,$$

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

We want to find an optimal portfolio $\mathbf{S}_t$ at each time t such that the expected terminal utility is maximized:

$$V_0(W_0) = \max_{\mathbf{S}_t, 0 \le t < T} \mathbb{E}\{u(W_T)\},$$

where $u(W) = W^{1-\gamma}/(1-\gamma)$ for some constant $\gamma > 0$ and $\gamma \neq 1$. Moreover, we assume that neither borrowing nor shorting is allowed (i.e., $B_t \ge 0$ and $\mathbf{S}_t \ge 0$ for all t).

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

$$\begin{aligned} V_t(W) = \max_{B,S \ge 0} \ & \mathbb{E}\{V_{t+1}(R_f B + \mathbf{R}^\top \mathbf{S})\}, \\ \text{s.t. } & W - B - \mathbf{e}^\top \mathbf{S} = 0, \end{aligned} \tag{9}$$

for $t = 0, 1, \ldots, T-1$, where W is the state variable, B and $\mathbf{S}$ are the control variables, and the terminal value function is $V_T(W) = u(W)$.[8] Here we assume that the stocks' one-period random return vector $\mathbf{R}$ has the identically independent distribution across all periods.

4.1.1 True solution

To determine the accuracy of the solutions from L-VFI and H-VFI, we use examples where we can find their true solutions without implementing value function iteration. Since the terminal utility function is $u(W) = W^{1-\gamma}/(1-\gamma)$, we know that $V_t(W) = \alpha_t W^{1-\gamma}$ with $\alpha_T = 1/(1-\gamma)$, and

$$\alpha_t = \mathbb{E}\left\{ \left(R_f \left(1 - \mathbf{e}^\top \mathbf{s}^*\right) + \mathbf{R}^\top \mathbf{s}^* \right)^{1-\gamma} \right\} \alpha_{t+1},$$

where $\mathbf{s}^*$ is the vector of the optimal allocation fractions of wealth invested in the stocks, for any $t < T$. This tells us that the optimal solutions are $B_t^* = \left(1 - \mathbf{e}^\top \mathbf{s}^*\right) W_t$ and $\mathbf{S}_t^* = \mathbf{s}^* W_t$ for all time t and any wealth W_t. Moreover, $\mathbf{s}^*$ can be easily obtained from computing the single-period portfolio optimization problem with the terminal power utility function. We use this as the true solution when we do our accuracy tests for L-VFI or H-VFI.

[8] This is a special case of the model (1) with $u_t \equiv 0$ for $t < T$ and $\beta = 1$ (the value of β does not matter if it is positive).

4.1.2 Bounded variant of normal random variable

In the dynamic portfolio optimization problem, a typical assumption about the stocks is that they have log-normal returns. This means that the next-stage value of any stock could be any number in $(0, \infty)$, making it difficult to approximate the value function well. Particularly, when the terminal utility function is the power utility with $\gamma > 1$, if wealth goes to 0, then the value function tends to $-\infty$. To overcome this unbounded problem, we assume that the stocks have bounded returns having distributions close to be log-normal in this example.

We use the following function to transform a standard normal random variable $\varsigma \sim \mathcal{N}(0, 1)$ to a bounded random variable Ψ:

$$\Psi = \frac{1 - e^{-\kappa\varsigma}}{1 + e^{-\kappa\varsigma}}\Upsilon, \tag{10}$$

where Υ and κ are two positive parameters. We see that Ψ has zero mean, and it is symmetric around the mean and bounded in $(-\Upsilon, +\Upsilon)$. Once we choose a number for Υ, we find a corresponding κ so that Ψ has a unit variance.[9] For example, if we set $\Upsilon = 4$, then we will let $\kappa = 0.532708$. Thus, Ψ is close to ς, particularly in the interval $(-2, 2)$ if we choose $\Upsilon \geq 4$.

Therefore, in this example, we assume that the stocks have bounded returns $\mathbf{R} = (R_1, \ldots, R_D)^\top$ with

$$\log(R_j) = \mu_j + \frac{1 - e^{-\kappa\varsigma_j}}{1 + e^{-\kappa\varsigma_j}}\Upsilon\sigma_j, \tag{11}$$

where ς_j is a standard normal random variable for $j = 1, \ldots, D$. We assume that the correlation matrix of $(\varsigma_1, \ldots, \varsigma_D)$ is Σ.

In the objective function of the Bellman equation (9), we need to compute the expectation of V_{t+1}. When we assume that the returns of the stocks are bounded with transformation from normal random variables, this expectation can be computed by the product Gaussian-Hermite quadrature (see "Appendix 3").

4.1.3 Nonlinear change of variable

When the relative risk aversion coefficient γ in the power utility function is bigger than 1, the value function is steep when the wealth is nearly 0, and it becomes very flat when the wealth is large. Thus, it will be hard to approximate the value function well with a polynomial of the state variable W. Therefore, to approximate the value function accurately, approximation nodes should be assigned in such a way that they are denser when they are closer to the small lower bound, and they are sparser when they are closer to the large upper bound.

[9] We provide a MATLAB subroutine in https://sites.google.com/site/dphermite/boundednormal to compute κ for any given $\Upsilon > 0$.

To solve this problem, we set $w = \log(W)$ instead of W as our state variable. We approximate the value function on an interval $\left(\underline{W}, \overline{W}\right)$ with

$$\hat{V}(W; \mathbf{b}) = \sum_{j=0}^{n} b_j \mathcal{T}_j \left(\frac{2\log(W) - \overline{w} - \underline{w}}{\overline{w} - \underline{w}} \right),$$

where $\underline{w} = \log(\underline{W})$ and $\overline{w} = \log(\overline{W})$, and $\mathcal{T}_j$ are the Chebyshev basis polynomials. Moreover, we choose the approximation nodes as

$$W_i = \exp\left(\frac{(z_i + 1)(\overline{w} - \underline{w})}{2} + \underline{w} \right)$$

with $z_i = -\cos\left((2i - 1)\pi/(2m)\right)$ for $i = 1, \ldots, m$. From our experience, this approach helps constructing accurate polynomial approximations of the value functions.

Since our example does not allow shorting or borrowing, and the i-th stock return is assumed to be bounded in $\left(e^{\mu_i - \Upsilon\sigma_i}, e^{\mu_i + \Upsilon\sigma_i}\right)$, we can obtain the approximation intervals $[\underline{W}_t, \overline{W}_t]$ recursively as

$$\begin{aligned} \underline{W}_{t+1} &= \min_{j=1,\ldots,D} \left\{ e^{\mu_j - \Upsilon\sigma_j} \right\} \underline{W}_t, \\ \overline{W}_{t+1} &= \max_{j=1,\ldots,D} \left\{ e^{\mu_j + \Upsilon\sigma_j} \right\} \overline{W}_t, \end{aligned}$$

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

4.1.4 Numerical examples

In our numerical examples, we choose a portfolio with one risk-less bond and $D = 4$ stocks having the bounded return $\mathbf{R} = (R_1, ..., R_4)^\top$ specified in (11). We use the distributions of the risky returns from Infanger (2006): the mean of $\log(\mathbf{R})$ is $\mu = (0.0956, 0.0897, 0.0878, 0.0778)^\top$, the standard deviation of $\log(\mathbf{R})$ is $\sigma = (0.1572, 0.1675, 0.0657, 0.0489)^\top$, and the correlation matrix of corresponding $(\varsigma_1, ..., \varsigma_4)$ in the formula (11) is

$$\Sigma = \begin{bmatrix} 1 & 0.601 & 0.247 & 0.062 \\ 0.601 & 1 & 0.125 & 0.027 \\ 0.247 & 0.125 & 1 & 0.883 \\ 0.062 & 0.027 & 0.883 & 1 \end{bmatrix}.$$

The interest rate of the risk-less bond is $r = 0.05$. We let the initial wealth bound be $[\underline{W}_0, \overline{W}_0] = [0.9, 1.1]$, and choose $\Upsilon = 4$ and $\kappa = 0.532708$ in the formula (11), and let $T = 6$. Thus, the terminal wealth bound becomes $[\underline{W}_6, \overline{W}_6] = [0.044, 66.11]$.

Table 1 Errors and running times of L-VFI or H-VFI for dynamic portfolio optimization

γ	m	L-VFI error	H-VFI error	L-VFI time (seconds)	H-VFI time (seconds)
0.5	5	7.2 (−3)	1.3 (−7)	11	23
	10	3.8 (−7)	1.2 (−6)	35	64
	20	1.0 (−9)		152	
2	5	5.4 (−2)	9.1 (−5)	15	23
	10	9.2 (−5)	6.6 (−6)	43	83
	20	8.5 (−7)		158	
5	10	8.9 (−1)	6.7 (−3)	40	85
	20	6.5 (−3)	1.5 (−6)	159	299
	40	1.3 (−6)		587	

The column "L-VFI error" lists the $\mathcal{L}^\infty$ absolute errors of optimal stock allocations of all stocks at the initial stage from L-VFI, and the column "L-VFI time" lists the running times of L-VFI. Similarly, "H-VFI error" and "H-VFI time" list the errors and times of H-VFI. $a(k)$ means $a \times 10^k$

The computational results of L-VFI or H-VFI are given by GAMS code, and the optimization solver in the maximization step of L-VFI or H-VFI is CONOPT (Drud 1996). We use the Chebyshev polynomial interpolation as the approximation method, and use the product Gaussian-Hermite quadrature formula with seven quadrature nodes in each dimension to compute the expectation in the objective function of the Bellman eq. (9).

Table 1 lists absolute errors of the initial optimal stock allocations and running times[10] of L-VFI or H-VFI for three relative risk aversion coefficients $\gamma = 0.5, 2, 5$, and various numbers of approximation nodes m.[11] From Table 1, we see that a higher γ will need a higher m to obtain small errors. When γ is too high (e.g., $\gamma = 10$), Chebyshev polynomials cannot approximate the value functions well with a reasonable m.

With the same m Chebyshev nodes, L-VFI uses the degree $m-1$ Chebyshev polynomial interpolation on the Lagrange data and its coefficients are computed by the regression algorithm (see "Appendix 1"); H-VFI uses the degree $2m-1$ Chebyshev polynomial interpolation on the Hermite data and its coefficients are computed by solving the linear system (7). From Table 1, we see that H-VFI obtains higher accuracy than L-VFI using the same approximation nodes (except the case with $\gamma = 0.5$ and $m = 10$, which has already had six digits accuracy). Moreover, if L-VFI wants to achieve the same accuracy of H-VFI using m approximation nodes, it should use about $2m$ approximation nodes, and will then take nearly double computation time.

[10] All the examples are run on a 2.5 GHz Intel core.

[11] We omit some cases of $m = 20$ or 40 for H-VFI to make it clearer in comparison between L-VFI and H-VFI. Moreover, for economics problems, usually the accuracy $O(10^{-6})$ is enough.

4.2 Multi-country optimal growth problems

The use of slope information is more important for multidimensional DP problems, where computing the optimal policy will be time-consuming for each approximation node. For a d-dimensional DP problem, H-VFI uses not only $V(\mathbf{x}_i)$ at m approximation nodes $\mathbf{x}_i$ but also $d \times m$ slopes which come almost freely at cost, while L-VFI can only use m values of $V(\mathbf{x}_i)$. Thus, H-VFI will reduce the computation time significantly, if one wants to achieve the same accuracy of L-VFI. We show this by solving multi-country optimal growth problems (Den Haan et al. 2011; Juillard and Villemot 2011).[12] Here we adapt the model to be a finite-horizon problem. This allows us to find its true solution by solving it directly as a large-scale optimal control problem. We then use the true solution for accuracy tests to L-VFI and H-VFI.

We assume that there are d countries, and let $\mathbf{k}_t = (k_{t,1}, \ldots, k_{t,d})$ denote the vector of capital stocks of these countries at time t. Let $\boldsymbol{\ell}_t = \left(\ell_{t,1}, \ldots, \ell_{t,d}\right)$ denote the vector of elastic labor supply levels of the countries. Let the net production of country j at time t be

$$f(k_{t,j}, \ell_{t,j}) = A k_{t,j}^{\psi} \ell_{t,j}^{1-\psi},$$

with $A = (1-\beta)/(\psi\beta)$, for $j = 1, \ldots, d$. Let $\mathbf{c}_t = \left(c_{t,1}, \ldots, c_{t,d}\right)$ denote the vector of consumptions of the countries. The utility function is

$$u(\mathbf{c}, \boldsymbol{\ell}) = \sum_{j=1}^{d} \left[\frac{\left(c_j/A\right)^{1-\gamma} - 1}{1-\gamma} - (1-\psi) \frac{\ell_j^{1+\eta} - 1}{1+\eta} \right].$$

We want to find the optimal consumption and labor supply decisions such that the expected total utility over a finite-horizon time is maximized. That is,

$$\begin{aligned} V_0(\mathbf{k}_0) = \max_{\mathbf{I}_t, \mathbf{c}_t, \boldsymbol{\ell}_t} & \sum_{t=0}^{T-1} \beta^t u(\mathbf{c}_t, \boldsymbol{\ell}_t) + \beta^T V_T(\mathbf{k}_T), \\ \text{s.t. } & k_{t+1,j} = (1-\delta) k_{t,j} + I_{t,j}, \quad j = 1, \ldots, d, \\ & \Gamma_{t,j} = \frac{\zeta}{2} k_{t,j} \left(\frac{I_{t,j}}{k_{t,j}} - \delta \right)^2, \quad j = 1, \ldots, d, \\ & \sum_{j=1}^{d} \left(c_{t,j} + I_{t,j} - \delta k_{t,j} \right) = \sum_{j=1}^{d} \left(f(k_{t,j}, \ell_{t,j}) - \Gamma_{t,j} \right), \end{aligned} \tag{12}$$

where δ is the depreciation rate of capital, $\mathbf{I}_t = (I_{t,1}, ..., I_{t,d})$ is the vector of investments of the countries, $\Gamma_{t,j}$ is the investment adjustment cost of country j, and ζ governs the intensity of the friction.

[12] Many papers tried to get an approximate solution to such a real-business cycle model using various methods including value function iteration (e.g., Trick and Zin 1997; Aldrich et al. 2011; Judd et al. 2011; Malin et al. 2011).

The DP formulation of the multi-country model (12) in H-VFI is

$$\begin{aligned} V_t(\mathbf{k}) = \max_{\mathbf{I},\mathbf{c},\boldsymbol{\ell},\mathbf{y}} \; & u(\mathbf{c},\boldsymbol{\ell}) + \beta V_{t+1}(\mathbf{k}^+), \\ \text{s.t. } & k_j^+ = (1-\delta)y_j + I_j, \quad j = 1,\ldots,d, \\ & \Gamma_j = \frac{\zeta}{2} y_j \left(\frac{I_j}{y_j} - \delta\right)^2, \quad j = 1,\ldots,d, \\ & \sum_{j=1}^{d} \left(c_j + I_j - \delta y_j\right) = \sum_{j=1}^{d} \left(f(y_j,\ell_j) - \Gamma_j\right), \\ & k_j - y_j = 0, \quad j = 1,\ldots,d, \end{aligned} \tag{13}$$

for $t < T$. The dummy variable y_j and the trivial constraint $k_j - y_j = 0$ are used in order to get the gradient of the value function directly from the optimization solver as described in H-VFI, for $j = 1,\ldots,d$.

In the examples of this subsection, we let $T = 5$, $\psi = 0.36$, $\delta = 0.025$, $\zeta = 0.5$, and let the domain of the capital stocks be $[0.5, 1.5]^d$. The parameter values come from Juillard and Villemot (2011). The terminal value function is $V_T(\mathbf{k}) = u(f(\mathbf{k},\mathbf{e}),\mathbf{e})/(1-\beta)$ where $\mathbf{e}$ is the d-dimensional vector of ones. We get the true solution of this multi-country model by solving (12) directly using the CONOPT solver in the GAMS environment.

We first solve a three-country RBC problem (i.e., $d = 3$) with various parameters. Figure 1 displays relative errors of the optimal consumption and labor supply, and running times [13] of L-VFI or H-VFI, for various $\beta = 0.9, 0.95, 0.99$, $\gamma = 0.5, 2, 5$, and $\eta = 0.2, 1, 5$.[14] We use tensor grid of $m = 5, 7, 10$ expanded Chebyshev nodes in each dimension (the overall number of nodes is m^d). For the Lagrange approximation using m^d nodes, we use degree $m-1$ complete Chebyshev polynomials and apply the regression algorithm (see "Appendix 1") to compute its coefficients. And for the Hermite approximation using m^d nodes, we use degree $2m-1$ complete Chebyshev polynomials and apply the least square method (8) to compute its coefficients. We implement L-VFI and H-VFI with GAMS code and use CONOPT as the optimization solver. In the figure, the stars, the marks, and the pluses represent the log10 of relative errors and the log10 of running times of L-VFI with $m = 5, 7, 10$, respectively, while the circles and the squares represent those of H-VFI with $m = 5$ and 7,[15] for various β, γ, and η.

Figure 1 shows that, for any combination of (β, γ, η), the solutions of H-VFI (circles in the figure) have more than two digits higher accuracy than the solutions of L-VFI (stars in the figure), using the same $5^3 = 125$ approximation nodes. Moreover, the running times of H-VFI (circles in the figure) are only around 120 s, just a bit more

[13] All the examples in this paper are run on a 2.5 GHz Intel core of a laptop.

[14] The ranges of β, γ, and η have already covered the most of values in the economics literature for the optimal growth problems.

[15] We omit the cases of H-VFI with $m = 10$ because H-VFI with $m = 7$ has produced solutions with the accuracy at $O(10^{-6})$.

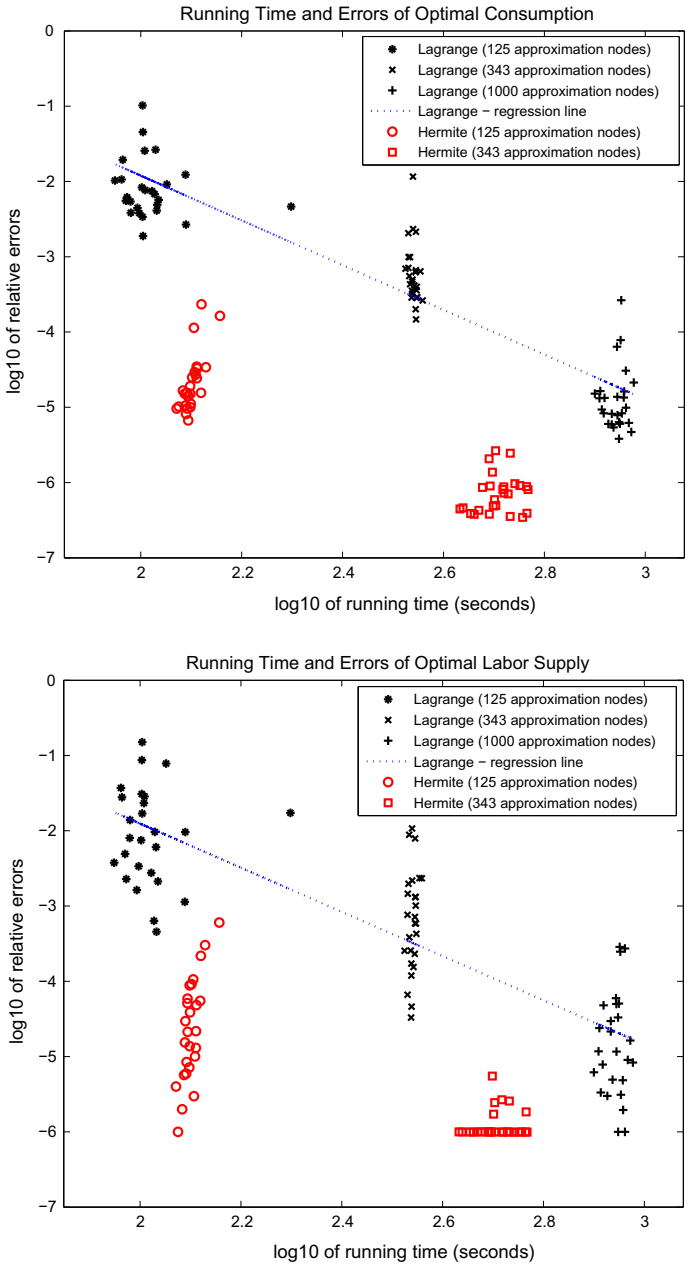

Fig. 1 L-VFI vs H-VFI for three-country optimal growth problems

than L-VFI (around 100 s). Figure 1 also shows that H-VFI with 125 approximation nodes (circles in the figure) is not only nearly three times faster but also about one digit more accurate than L-VFI with $7^3 = 343$ approximation nodes (marks in the figure).

Figure 1 also tells us that if L-VFI wants to have the same accuracy as H-VFI with 125 approximation nodes (circles in the figure), it uses 1000 approximation nodes (pluses in the figure) and takes about seven times the computation time of H-VFI, for the three-country optimal growth problems. Moreover, H-VFI with $m = 7$ (squares in the figure) can achieve the higher accuracy up to $O(10^{-6})$ using less computation time than L-VFI with $m = 10$ (pluses in the figure). The results are more clearly shown in Table 2, which lists the relative errors of the optimal consumption and the running times of several selected cases.

From Fig. 1 and Table 2, we see that the higher efficiency of H-VFI relative to L-VFI is almost the same for all parameter choices. Next, we solve the multi-country RBC problem with $\beta = 0.99$, $\gamma = 5$, and $\eta = 5$, and we use different numbers of countries to check the relationship of the efficiency of H-VFI and the dimension of continuous states. Table 3 lists relative errors of the optimal consumption and labor supply, and running times of L-VFI or H-VFI for $d = 3$, 4, and 5 countries, using

Table 2 Errors and running times of L-VFI or H-VFI for three-country growth problems

β	γ	η	m	L-VFI error	H-VFI error	L-VFI time (seconds)	H-VFI time (seconds)
0.9	0.5	0.2	5	3.8 (−3)	9.6 (−6)	99	124
			7	2.9 (−4)	4.3 (−7)	344	468
			10	5.4 (−6)		865	
0.95	2	1	5	5.4 (−3)	1.4 (−5)	95	124
			7	4.0 (−4)	9.0 (−7)	353	493
			10	7.8 (−6)		881	
0.99	5	5	5	1.0 (−1)	2.3 (−4)	101	132
			7	1.2 (−2)	2.6 (−6)	346	505
			10	2.6 (−4)		896	

$a(k)$ means $a \times 10^k$

Table 3 Errors and running times of L-VFI or H-VFI for multi-country growth problems

d	m	Error of c_0^*		Error of l_0^*		Time (minutes)	
		L-VFI	H-VFI	L-VFI	H-VFI	L-VFI	H-VFI
3	4	1.1 (−1)	2.5 (−3)	1.1(−1)	2.4 (−3)	1.1	2.2
	7	1.2 (−2)	2.6 (−6)	1.1 (−2)	2.4 (−6)	5.8	8.4
	8	3.7 (−3)		3.4 (−3)		9.1	
4	4	9.2 (−2)	2.3 (−3)	9.4 (−2)	2.2 (−3)	3.4	4.3
	7	2.3 (−3)	7.0 (−7)	2.2 (−3)	1.8 (−6)	36.8	188.6
5	4	1.0 (−1)	2.4 (−3)	1.0 (−1)	2.2 (−3)	14.0	31.5
	5	4.7 (−2)	2.2 (−4)	4.2 (−2)	2.1 (−4)	45.8	241.1
	7	2.2 (−3)		2.0 (−3)		349.1	

$a(k)$ means $a \times 10^k$

tensor grid of m expanded Chebyshev nodes in each dimension. We see that when $m = 7$, H-VFI achieves the accuracy up to $O(10^{-6})$ for $d = 3$ or 4,[16] while L-VFI can have only $O(10^{-2})$ or $O(10^{-3})$. Moreover, from the table, to achieve the same accuracy of H-VFI with $m = 4$, L-VFI needs 4.1 times the computation time of H-VFI for the three-country example, 8.6 times for the four-country example, and 11.1 times for the five-country example. Therefore, we conclude that the speed advantage of H-VFI over L-VFI increases with the dimension of continuous states.

4.3 Six-country optimal stochastic growth problems

In this subsection, we show the advantage of H-VFI with six-country stochastic growth problems (12). The setting is similar to the previous examples, but we assume that the net productions of all countries are dependent on a common random economic shock θ_t, which is a Markov chain. That is, the net production of country j at time t is

$$f(k_{t,j}, \ell_{t,j}, \theta_t) = \theta_t A k_{t,j}^{\psi} \ell_{t,j}^{1-\psi},$$

for $j = 1, ..., d$ with $d = 6$. The possible values of θ_t are $\vartheta_1 = 0.98$ and $\vartheta_2 = 1.02$, and the probability transition matrix from θ_t to $\theta_{t+1} = g(\theta_t, \epsilon_t)$ is

$$\begin{bmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{bmatrix},$$

for $t = 0, \ldots, T-1$. Therefore, we have the stochastic version of the multi-country model (12):

$$\begin{aligned} V_0(\mathbf{k}_0, \theta_0) = \max_{\mathbf{I}_t, \mathbf{c}_t, \boldsymbol{\ell}_t} \ & \mathbb{E}\left\{ \sum_{t=0}^{T-1} \beta^t u(\mathbf{c}_t, \boldsymbol{\ell}_t) + \beta^T V_T(\mathbf{k}_T, \theta_T) \right\}, \\ \text{s.t. } & k_{t+1,j} = (1-\delta) k_{t,j} + I_{t,j}, \quad j = 1, \ldots, d, \\ & \Gamma_{t,j} = \frac{\zeta}{2} k_{t,j} \left(\frac{I_{t,j}}{k_{t,j}} - \delta \right)^2, \quad j = 1, \ldots, d, \\ & \sum_{j=1}^{d} \left(c_{t,j} + I_{t,j} - \delta k_{t,j} \right) = \sum_{j=1}^{d} \left(f(k_{t,j}, \ell_{t,j}, \theta_t) - \Gamma_{t,j} \right), \\ & \theta_{t+1} = g(\theta_t, \epsilon_t). \end{aligned} \tag{14}$$

In the following examples, the number of the decision stages is $T = 5$, and the domain of the capitals is $[0.5, 1.5]^6$. We apply the tree method (see "Appendix 4") to compute the "true" solutions at test points of $\mathbf{k}_0$ and each possible value of θ_0. We implement L-VFI and H-VFI with GAMS code and use CONOPT as the optimization

[16] We omit the case of H-VFI with $d = 5$ and $m = 7$, because it will be too time-consuming to run the GAMS code using a laptop. But for the five-country problem, H-VFI with $m = 5$ achieves $O(10^{-4})$ accuracy, which is one-digit more accurate and also faster than L-VFI with $m = 7$.

Table 4 H-VFI versus L-VFI for six-dimensional stochastic problems

m	Error of c_0^*		Error of l_0^*		Running times (hour)	
	L-VFI	H-VFI	L-VFI	H-VFI	L-VFI	H-VFI
Case 1: $(\beta, \gamma, \eta) = (0.95, 2, 1)$						
3	4.3 (−2)	2.0 (−3)	6.0 (−2)	3.0 (−3)	0.3	0.66
4	1.2 (−2)	1.1 (−4)	1.7 (−2)	1.6 (−4)	2.0	14.3
5	3.2 (−3)		4.7 (−3)		8.7	
6	9.3 (−4)		1.4 (−3)		36.6	
Case 2: $(\beta, \gamma, \eta) = (0.99, 5, 5)$						
3	2.9 (−1)	1.1 (−2)	2.1 (−1)	1.0 (−2)	0.3	0.8
4	6.6 (−2)	1.5 (−3)	6.6 (−2)	1.4 (−3)	2.1	17.9
5	1.7 (−2)		1.5 (−2)		10.5	
6	3.8 (−3)		3.5 (−3)		47.9	

$a(k)$ means $a \times 10^k$

solver. Since it is time-consuming to run such examples with the GAMS code using a laptop,[17] here we just give two cases of $(\beta, \gamma, \eta) = (0.95, 2, 1)$ and $(0.99, 5, 5)$, and run H-VFI with $m = 3$ or 4 only.

Table 4 lists running times and relative errors of the optimal consumption and labor supply of L-VFI and H-VFI using tensor grid of m expanded Chebyshev nodes in each dimension. On the tensor grid of m nodes in each dimension, L-VFI uses degree $m-1$ complete Chebyshev polynomials, and H-VFI uses degree $2m-1$ complete Chebyshev polynomials and the least squares model (8) is applied to obtain its coefficients.

From Table 4, we see that for the first case $(\beta, \gamma, \eta) = (0.95, 2, 1)$, H-VFI using $m^d = 3^6$ approximation nodes takes only 0.66 h to achieve a higher accuracy than L-VFI using 5^6 (i.e., $m = 5$) approximation nodes which takes 8.7 h, about 13 times the computation time of H-VFI. Moreover, if L-VFI wants to use 6^6 (i.e., $m = 6$) approximation nodes to achieve a bit higher accuracy than H-VFI with $m = 3$, its running time is up to 36.6 h, about 55 times the computation time of H-VFI. But H-VFI with $m = 4$ is about 2.5 times faster and also nearly one-digit more accurate than L-VFI with $m = 6$. The second case $(\beta, \gamma, \eta) = (0.99, 5, 5)$ has the similar results. It shows again that the efficiency of H-VFI relative to L-VFI is almost independent of the values of (β, γ, η), which has already been shown in Fig. 1 and Table 2.

5 Conclusion

We have shown that gradients of value functions can be obtained easily and almost freely during value function iteration, and using them in Hermite approximation methods will produce significant improvement in the efficiency of numerical dynamic

[17] All the running times can be dramatically reduced if we use Fortran code and the NPSOL solver (Gill et al. 1994), but the relative difference between H-VFI and L-VFI is still almost the same.

programming. These conclusions are supported by examples from dynamic portfolio optimization problems and multi-dimensional optimal growth problems. We have shown that this improvement of H-VFI over L-VFI can be used to either increase the accuracy within a given amount of computation time, or attain the same accuracy with much less computation time. These examples also indicate that H-VFI will be particularly valuable in solving multi-dimensional dynamic programming problems.

Appendix 1: One-dimensional Chebyshev polynomial approximation

One-dimensional Chebyshev basis polynomials on $[-1, 1]$ are defined as $\mathcal{T}_j(x) = \cos(j \cos^{-1}(x))$, while general Chebyshev polynomials on $[x_{\min}, x_{\max}]$ are defined as $\mathcal{T}_j(Z(x))$ where

$$Z(x) = (2x - x_{\min} - x_{\max})/(x_{\max} - x_{\min})$$

for $j = 0, 1, 2, \ldots$. These polynomials are orthogonal under the weighted inner product: $\langle f, g \rangle = \int_{x_{\min}}^{x_{\max}} f(x)g(x)w(x)dx$ with the weighting function $w(x) = \left(1 - (Z(x))^2\right)^{-1/2}$. The degree n Chebyshev polynomial approximation for $V(x)$ on $[x_{\min}, x_{\max}]$ is

$$\hat{V}(x; \mathbf{b}) = \sum_{j=0}^{n} b_j \mathcal{T}_j(Z(x)), \tag{15}$$

where $\mathbf{b} = \{b_j\}$ are the Chebyshev coefficients.

If we choose the Chebyshev nodes on $[x_{\min}, x_{\max}]$: $x_i = (z_i + 1)(x_{\max} - x_{\min})/2 + x_{\min}$ with $z_i = -\cos((2i - 1)\pi/(2m))$ for $i = 1, \ldots, m$, and Lagrange data $\{(x_i, v_i) : i = 1, \ldots, m\}$ are given (where $v_i = V(x_i)$), then the coefficients $\{b_j\}$ in (15) can be easily computed by the following formula:

$$\begin{aligned} b_0 &= \frac{1}{m} \sum_{i=1}^{m} v_i, \\ b_j &= \frac{2}{m} \sum_{i=1}^{m} v_i \mathcal{T}_j(z_i) \qquad j = 1, \ldots, n. \end{aligned} \tag{16}$$

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

When the number of the Chebyshev nodes is equal to the number of the Chebyshev coefficients (i.e., $m = n+1$), the approximation (15) with the coefficients given by (16) becomes the Chebyshev polynomial interpolation (which is a Lagrange interpolation), as $\hat{V}(x_i; \mathbf{b}) = v_i$, for $i = 1, \ldots, m$.

It is often more stable to use the expanded Chebyshev polynomial interpolation (Cai et al. 2010), as the above standard Chebyshev polynomial interpolation gives poor approximation in the neighborhood of the end points of the approximation interval. That is, we use the following formula to approximate $V(x)$:

$$\hat{V}(x;\mathbf{b}) = \sum_{j=0}^{n} b_j \mathcal{T}_j\left(\frac{2x - \widetilde{x}_{\min} - \widetilde{x}_{\max}}{\widetilde{x}_{\max} - \widetilde{x}_{\min}}\right), \tag{17}$$

where $\widetilde{x}_{\min} = x_{\min} - \delta$ and $\widetilde{x}_{\max} = x_{\max} + \delta$ with $\delta = (z_1 + 1)(x_{\min} - x_{\max})/(2z_1)$. Moreover, if we choose the expanded Chebyshev nodes on $[x_{\min}, x_{\max}]$: $x_i = (z_i + 1)(\tilde{x}_{\max} - \tilde{x}_{\min})/2 + \tilde{x}_{\min}$, then the coefficients $\{b_j\}$ can also be calculated easily by the expanded Chebyshev regression algorithm (Cai et al. 2010), which is similar to (16).

Appendix 2: Multidimensional complete Chebyshev polynomial approximation

In a d-dimensional approximation problem, let the approximation domain of the value function be

$$\left\{\mathbf{x} = (x_1, \ldots, x_d) : x_{\min,j} \le x_j \le x_{\max,j},\ j = 1, \ldots d\right\},$$

for some real numbers $x_{\min,j}$ and $x_{\max,j}$ with $x_{\max,j} > x_{\min,j}$ for $j = 1, \ldots, d$. Let $\mathbf{x}_{\min} = (x_{\min,1}, \ldots, x_{\min,d})$ and $\mathbf{x}_{\max} = (x_{\max,1}, \ldots, x_{\max,d})$. Then we denote $[\mathbf{x}_{\min}, \mathbf{x}_{\max}]$ as the approximation domain. Let $\alpha = (\alpha_1, \ldots, \alpha_d)$ be a vector of non-negative integers. Let $\mathcal{T}_\alpha(\mathbf{z})$ denote the product $\prod_{1 \le j \le d} \mathcal{T}_{\alpha_j}(z_j)$ for $\mathbf{z} = (z_1, \ldots, z_d) \in [-1, 1]^d$. Let

$$\mathbf{Z}(\mathbf{x}) = \left(\frac{2x_1 - x_{\min,1} - x_{\max,1}}{x_{\max,1} - x_{\min,1}}, \ldots, \frac{2x_d - x_{\min,d} - x_{\max,d}}{x_{\max,d} - x_{\min,d}}\right)$$

for any $\mathbf{x} = (x_1, \ldots, x_d) \in [\mathbf{x}_{\min}, \mathbf{x}_{\max}]$.

Using these notations, the degree n complete Chebyshev approximation for $V(\mathbf{x})$ is

$$\hat{V}_n(\mathbf{x};\mathbf{b}) = \sum_{0 \le |\alpha| \le n} b_\alpha \mathcal{T}_\alpha\left(\mathbf{Z}(\mathbf{x})\right), \tag{18}$$

where $|\alpha| = \sum_{j=1}^{d} \alpha_j$. The number of terms with $0 \le |\alpha| = \sum_{j=1}^{d} \alpha_i \le n$ is $\binom{n+d}{d}$ for the degree n complete Chebyshev approximation in $\mathbb{R}^d$.

Appendix 3: Multivariate numerical integration

From the formula (11), we know that $V_{t+1}(R_f B + \mathbf{R}^\top \mathbf{S})$ in the objective function of (9) can be rewritten as $G(\varsigma)$ for some function G where ς is the D-dimensional vector of the corresponding standard normal random variables with a correlation matrix Σ. Since Σ must be a positive semi-definite matrix from the covariance property, we can have its Cholesky factorization, $\Sigma = LL^\top$, where L is a $D \times D$ lower triangular matrix. Therefore,

$$
\begin{aligned}
&\mathbb{E}\left\{V_{t+1}(R_f B + \mathbf{R}^\top \mathbf{S})\right\} = \mathbb{E}\{G(\varsigma)\} \\
&= \left((2\pi)^D \det(\Sigma)\right)^{-1/2} \int_{\mathbb{R}^D} G(\mathbf{q}) e^{-\mathbf{q}^\top \Sigma^{-1} \mathbf{q}/2} d\mathbf{q} \\
&= \left((2\pi)^D \det(L)^2\right)^{-1/2} \int_{\mathbb{R}^D} G\left(\sqrt{2} L\mathbf{q}\right) e^{-\mathbf{q}^\top \mathbf{q}} \cdot 2^{D/2} \det(L) d\mathbf{q} \\
&\doteq \pi^{-\frac{D}{2}} \sum_{i_1=1}^{K} \cdots \sum_{i_D=1}^{K} w_{i_1} \cdots w_{i_D} G\left(\left(\sum_{j=1}^{D} L_{D,j} q_{i_j}\right) \sqrt{2} L_{1,1} q_{i_1}, \right. \\
&\left. \sqrt{2}(L_{2,1} q_{i_1} + L_{2,2} q_{i_2}), \cdots, \sqrt{2}\left(\sum_{j=1}^{D} L_{D,j} q_{i_j}\right)\right),
\end{aligned}
$$

where $\{w_i : 1 \le i \le K\}$ and $\{q_i : 1 \le i \le K\}$ are the Gauss-Hermite quadrature weights and nodes over $(-\infty, \infty)$, $L_{i,j}$ is the (i, j)-element of L, and $\det(\cdot)$ means the matrix determinant operator.

Appendix 4: Tree method

Assume that θ_t is a Markov chain with possible states, $\vartheta_1, \ldots, \vartheta_M$, and the probability of going from state ϑ_i to state ϑ_j in one step is

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

Thus, from time 0 to time t, there are M^t paths of θ_t for a given initial state, for $0 \le t \le T$. These paths comprise a tree structure. We can then compute the exact solutions of the dynamic stochastic problems with the tree structure by applying an optimization solver to solve a large-scale optimal control problem directly, called the tree method here.

In the tree method for the stochastic optimal growth problem (14), the goal is to find the optimal decisions $\left\{\left(\mathbf{I}_{t,i}, \mathbf{c}_{t,i}, \boldsymbol{\ell}_{t,i}\right) : 1 \le i \le M^t,\ 0 \le t < T\right\}$ to maximize the expected total utility:

$$
\begin{aligned}
\max_{\mathbf{I},\mathbf{c},\boldsymbol{\ell}} \quad & \sum_{t=0}^{T-1} \beta^t \sum_{i=1}^{M^t} (P_{t,i} u(\mathbf{c}_{t,i}, \boldsymbol{\ell}_{t,i})) + \\
& \beta^T \sum_{i=1}^{M^{T-1}} P_{T-1,i} \sum_{j=1}^{M} p_{\mathrm{mod}(i-1,M)+1,j} V_T(\mathbf{k}_{T,i}, \vartheta_j),
\end{aligned} \tag{19}
$$

subject to the corresponding constraints of (14) for all paths, where $\mathrm{mod}(i-1, M)$ is the remainder of division of $(i-1)$ by M, $P_{t,i}$ is the probability of the i-th path from time 0 to time t with the following recursive formula:

$$P_{t+1,(i-1)M+j} = P_{t,i} \cdot p_{\mathrm{mod}(i-1,M)+1,j},$$

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

This approach is practical for only small values of M and T. In our examples, we set $M = 2$ and $T = 5$, implying that there are only 32 possible paths, and nonlinear optimization solvers will produce results with as much precision as possible in a double precision environment. We will treat those solutions as the "true" solution which can be used for the error analysis of numerical DP algorithms.

References

Aldrich EM, Fernandez-Villaverde J, Gallant AR, Rubio- Ramrez JF (2011) Tapping the supercomputer under your desk: solving dynamic equilibrium models with graphics processors. J Econ Dyn Control 35:386–393

Bellman R (1957) Dynamic programming. Princeton University Press, Princeton

Bertsekas D (2005) Dynamic programming and optimal control, vol 1. Athena Scientific, Belmont

Bertsekas D (2007) Dynamic programming and optimal control, vol 2. Athena Scientific, Belmont

Bisschop JJ, Meeraus A (1982) On the development of a general algebraic modeling system in a strategic planning environment. Math Program Stud 20:1–29

Brooke A, Kendrick D, Meeraus A, Raman R (1997) GAMS language guide. Technical report, GAMS Development Corporation

Byrd RH, Nocedal J, Waltz RA (2006) KNITRO: an integrated package for nonlinear optimization. http://www.ziena.com/papers/integratedpackage. Technical report, Ziena

Cai Y (2010) Dynamic programming and its application in economics and finance. PhD thesis, Stanford University

Cai Y, Judd KL (2012) Dynamic programming with shape-preserving rational spline Hermite interpolation. Econ Lett 117(1):161–164

Cai Y, Judd KL (2013) Shape-preserving dynamic programming. Math Methods Oper Res 77(3):407–421

Cai Y, Judd KL (2014) Advances in numerical dynamic programming and new applications. Chapter 8. In: Schmedders Karl, Judd Kenneth L (eds) Handbook of computational economics, vol 3. Elsevier, Amsterdam

Cai Y, Judd KL, Lontzek TS (2013) The social cost of stochastic and irreversible climate change. NBER working paper 18704

Cai Y, Judd KL, Xu R (2013) Numerical solution of dynamic portfolio optimization with transaction costs. NBER working paper No. 18709

Cai Y, Judd KL, Thain G, Wright S (2015) Solving dynamic programming problems on computational grid. Comput Econ 45(2):261–284

Cocco JF, Gomes FJ, Maenhout PJ (2005) Consumption and portfolio choice over the life cycle. Rev Financ Stud 18(2):491–533

Czyzyk J, Mesnier MP, Moré JJ (1998) The NEOS server. IEEE Comput Sci Eng 5:68–75

Den Haan WJ, Judd KL, Juillard M (2011) Computational suite of models with heterogeneous agents II:multi-country real business cycle models. J Econ Dyn Control 35:175–177

Dolan ED, Fourer R, Goux JP, Munson TS, Sarich J (2008) Kestrel: an interface from optimization modeling systems to the NEOS server. INFORMS J Comput 20(4):525–538

Drud AS (1996) CONOPT: a system for large scale nonlinear optimization. ARKI Consulting and Development A/S, Bagsvaerd

Fourer R, Gay DM, Kernighan BW (1990) A modeling language for mathematical programming. Manag Sci 36:519–554

Fourer R, Gay DM, Kernighan BW (2003) AMPL: a modeling language for mathematical programming, 2nd edn. Duxbury Press, Pacific Grove

Gill P, Murray W, Saunders MA, Wright MH (1994) User's Guide for NPSOL 5.0: a Fortran Package for Nonlinear Programming. Technical report, SOL, Stanford University

Gill P, Murray W, Saunders MA (2005) SNOPT: an SQP algorithm for largescale constrained optimization. SIAM Rev 47(1):99–131

Griebel M, Wozniakowski H (2006) On the optimal convergence rate of universal and nonuniversal algorithms for multivariate integration and approximation. Math Comput 75(255):1259–1286
Gropp W, Moré JJ (1997) Optimization environments and the NEOS server. In: Buhmann MD, Iserles A (eds) Approximation theory and optimization: tributes to M. J. D. Powell. Cambridge University Press, Cambridge, UK, pp 167–182
Gupta A, Murray W (2005) A framework algorithm to compute optimal asset allocation for retirement with behavioral utilities. Comput Optim Appl 32(1/2):91–113
Infanger G (2006) Dynamic asset allocation strategies using a stochastic dynamic programming approach. Chapter 5. In: Zenios SA, Ziemba WT (Eds) Handbook of asset and liability management, volume 1, North Holland
Judd KL (1998) Numerical methods in economics. The MIT Press, Cambridge
Judd KL, Maliar L, Maliar S (2011) Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models. Quant Econ 2:173–210
Juillard M, Villemot S (2011) Multi-country real business cycle models: accuracy tests and test bench. J Econ Dyn Control 35:178–185
Malin BA, Krueger D, Kubler F (2011) Solving the multi-country real business cycle model using a Smolyak-collocation method. J Econ Dyn Control 35:229–239
McCarl B, et al. (2011) McCarl expanded GAMS user guide, version 23.6. http://www.gams.com/mccarl/mccarlhtml/. Accessed 06 Sept 2012
Murtagh BA, Saunders MA (2003) MINOS 5.51 User's Guide. http://www.stanford.edu/group/SOL/guides/minos551. Technical report, SOL, Stanford University
Philbrick CR, Kitanidis PK (2001) Improved dynamic programming methods for optimal control of lumped parameter stochastic systems. Oper Res 49(3):398–412
Powell WB (2007) Approximate dynamic programming: solving the curses of dimensionality. Wiley Interscience, Hoboken
Powell WB, Van Roy B (2004) Approximate dynamic programming for high-dimensional dynamic resource allocation problems. Chapter 10. In: Si J, Barto AG, Powell W, Wunsch D (Eds) Handbook of learning and approximate dynamic programming. Wiley-IEEE Press, Hoboken, NJ
Rivlin TJ (1990) Chebyshev polynomials: from approximation theory to algebra and number theory. Wiley Interscience, New York
Rust J (1997) Using randomization to break the curse of dimensionality. Econometrica 65(3):487–516
Rust J (2008) Dynamic programming. In: Durlauf Steven N, Blume Lawrence E (eds) New palgrave dictionary of economics, 2nd edn. Palgrave Macmillan, Basingstoke
Rust J, Traub JF, Wozniakowski H (2002) Is there a curse of dimensionality for contraction fixed points in the worst case? Econometrica 70(1):285–329
Schumaker L (1983) On shape-preserving quadratic spline interpolation. SIAM J Numer Anal 20:854–864
Trick MA, Zin SE (1997) Spline approximations to value functions–linear programming approach. Macroecon Dyn 1:255–277
Wang SP, Judd KL (2000) Solving a savings allocation problem by numerical dynamic programming with shape-preserving interpolation. Comput Oper Res 27(5):399–408