Dynamic Programming with Hermite Approximation[☆]

Yongyang Cai[*]

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

Kenneth L. Judd

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

Abstract

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

Keywords: Numerical dynamic programming, value function iteration, Hermite approximation, multi-dimensional approximation, dynamic portfolio optimization, multi-country optimal growth model
JEL Classification: C61, C63

[☆]Cai and Judd gratefully acknowledge NSF support (SES-0951576).

[*]The corresponding is Dr. Yongyang Cai, Hoover Institution, 424 Galvez Mall, Stanford University, Stanford, CA, 94305. Phone: 650-862-6478; E-mail: yycai@stanford.edu.

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

1. Introduction

Dynamic optimization problems in economics are numerically challenging due to nonlinearities and/or dimensionality. Dynamic programming (DP) is the standard approach for dynamic optimization problems. In many economics problems, the state and control variables are continuous, and theory implies that the value function is continuously differentiable. The value function must be approximated numerically, but imperfectly since computers cannot represent arbitrary continuous functions. DP problems are often solved by value function iteration, where the period t value function is computed from the period $t+1$ value function, and the value function is known at the terminal time T.

DP algorithms typically use value function iteration, where each step generates Lagrange data to approximate a value function, where the Lagrange data comes from solving the optimization problem in a Bellman equation at many points in the state space. For the purposes of this paper, we call that approach L-VFI for Lagrange value function iteration. However, the Envelope Theorem tells us that the output of each such optimization problem can include the gradient of the value function at each point at essentially no cost. This paper proposes using both the Lagrange and slope information in Hermite approximation methods for constructing value functions in value function iteration; we call this approach H-VFI, Hermite value function iteration. First, we show how to write an optimization problem so that the gradient of the value function is the value of the shadow prices of some constraints. Second, we apply Hermite value function iteration to some common economics problems. In our one-dimensional examples, H-VFI improves accuracy by almost the same as if we doubled the number of points used in L-VFI. This corresponds to the simple intuition that knowing the value and slope of a function at a point is equivalent to knowing the value of the function at two points. In our three- and six-dimensional examples, we find that Hermite approximation often achieves the same accuracy as Lagrange approximation but at significantly less cost.

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

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

Numerical solutions to a DP problem are based on the Bellman equation (Bellman, 1957):

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

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

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

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

If state and control variables in a DP problem are continuous, then the value function must be approximated in some computationally tractable manner. It is common to approximate value functions with a finitely parameterized collection of functions; that is, we use some functional form $\hat{V}(x;\mathbf{b})$, where $\mathbf{b}$ is a vector of parameters, and approximate a value function, $V(x)$, with $\hat{V}(x;\mathbf{b})$ 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 fixed, we focus on finding the vector of parameters, $\mathbf{b}$, such that $\hat{V}(x;\mathbf{b})$ approximately satisfies the Bellman equation. 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, $\mathbb{X}_t = \{x_t^i : 1 \leq i \leq N_t\} \subset \mathbb{R}^d$, for every $t < T$, and choose a functional form for $\hat{V}(x;\mathbf{b})$. Let $\hat{V}(x;\mathbf{b}^T) \equiv V_T(x)$. Then for $t = T-1, T-2, \ldots, 0$, iterate through steps 1 and 2.*

Step *1. Maximization Step. Compute*

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

for each $x^i \in \mathbb{X}_t$, $1 \leq i \leq N_t$.

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.*

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

An approximation scheme consists of two parts: basis functions and approximation nodes. Approximation nodes can be chosen as uniformly spaced nodes, Chebyshev nodes, or some other specified nodes. From the viewpoint of basis functions, approximation methods can be classified as either spectral methods or finite element methods. A spectral method uses globally nonzero basis functions $\phi_j(x)$ such that $\hat{V}(x;\mathbf{b}) = \sum_{j=0}^{n} b_j \phi_j(x)$ is a degree-n approximation. Examples of spectral methods include ordinary polynomial approximation, Chebyshev polynomial approximation, and shape-preserving Chebyshev polynomial approximation (Cai and Judd, 2012b). In contrast, a finite element method uses locally basis functions $\phi_j(x)$ that are nonzero over sub-domains of the approximation domain. Examples of finite element methods include piecewise linear interpolation, cubic splines, and B-splines. See Cai (2009), Cai and Judd (2010), and Judd (1998) for more details.

2.1. Chebyshev Polynomial Approximation

Chebyshev 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 $\mathcal{T}_j((2x-x_{\min}-x_{\max})/(x_{\max}-x_{\min}))$ 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 $w(x)=\left(1-((2x-x_{\min}-x_{\max})/(x_{\max}-x_{\min}))^2\right)^{-1/2}$. A degree n Chebyshev polynomial approximation for $V(x)$ on $[x_{\min},x_{\max}]$ is

$$\hat{V}(x;\mathbf{b})=\sum_{j=0}^{n}b_j\mathcal{T}_j\left(\frac{2x-x_{\min}-x_{\max}}{x_{\max}-x_{\min}}\right), \tag{3}$$

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 $z_i=-\cos\left((2i-1)\pi/(2m)\right)$ 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 (3) can be easily computed by the following formula,

$$\begin{aligned} b_0 &= \frac{1}{m}\sum_{i=1}^{m}v_i, \\ b_j &= \frac{2}{m}\sum_{i=1}^{m}v_i\mathcal{T}_j(z_i), \qquad j=1,\ldots,n. \end{aligned} \tag{4}$$

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

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

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

$$\hat{V}(x;\mathbf{b})=\sum_{j=0}^{n}b_j\mathcal{T}_j\left(\frac{2x-\widetilde{x}_{\min}-\widetilde{x}_{\max}}{\widetilde{x}_{\max}-\widetilde{x}_{\min}}\right), \tag{5}$$

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, 2009), which is similar to (4).

2.2. Multidimensional Complete Chebyshev Polynomial Approximation

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

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

for some real numbers $x_j^{\min}$ and $x_j^{\max}$ with $x_j^{\max}>x_j^{\min}$ for $j=1,\ldots,d$. Let $x^{\min}=(x_1^{\min},\ldots,x_d^{\min})$ and $x^{\max}=(x_1^{\max},\ldots,x_d^{\max})$. Then we denote $[x^{\min},x^{\max}]$ as the domain. Let $\alpha=(\alpha_1,\ldots,\alpha_d)$ be a vector of nonnegative integers. Let $\mathcal{T}_\alpha(z)$ denote the product $\mathcal{T}_{\alpha_1}(z_1)\cdots\mathcal{T}_{\alpha_d}(z_d)$ for $z=(z_1,\ldots,z_d)\in[-1,1]^d$. Let

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

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

Using these notations, the degree-n complete Chebyshev approximation for $V(x)$ is

$$\hat{V}_n(x;\mathbf{b})=\sum_{0\le|\alpha|\le n} b_\alpha\mathcal{T}_\alpha\left(Z(x)\right), \tag{6}$$

where $|\alpha|=\sum_{j=1}^d\alpha_j$ for the nonnegative integer vector $\alpha=(\alpha_1,\ldots,\alpha_d)$. So 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 $\mathbb{R}^d$.

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 can produce the gradient of the value function at an approximation node as well as the value. Then we show how to use that information to produce better value function approximations.

3.1. H-VFI

Traditional approximation methods in DP problems use only Lagrange data. The Maximization Step in L-VFI is

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

for each pre-specified approximation node x^i, $i = 1,\dots,N$. Then it uses the Lagrange data set $\{(x^i, v_i) : i = 1,\dots,N\}$ in the Fitting Step to construct an approximation $\hat{V}_t(x)$ of the value function.

However, the Maximization Step of L-VFI can give not only the value function at an approximation node, but also its gradients, and do so at almost no cost, for DP problems with continuous and derivable value functions. This is accomplished by invoking the Envelope Theorem, which we next state.

Theorem 1 (Envelope Theorem). *Let*

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

where $x \in \mathbb{R}^d$. Suppose that $a^*(x)$ is the optimizer of (7), and that $\lambda^*(x)$ is the vector of shadow prices for the equality constraints $g(x,a) = 0$, and $\mu^*(x)$ is the vector of shadow prices of the inequality constraints $h(x,a) = 0$,. Then

$$\frac{\partial H(x)}{\partial x_j} = \frac{\partial f}{\partial x_j}(x,a^*(x)) + \lambda^*(x)^\top \frac{\partial g}{\partial x_j}(x,a^*(x)) + \mu^*(x)^\top \frac{\partial h}{\partial x_j}(x,a^*(x)), \tag{8}$$

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

Formula (8) expresses the value of $\partial H(x)/\partial x_j$ in terms of the shadow prices, objective gradient, constraints gradients at the optimum. The key simplifying feature of (8) is the absence of any term expressing how the choice of a is affected by a local change in state.

We could use Formula (8) to generate Hermite data for a value function, but there is a better way to compute those gradients. Corollary 1 shows how to rewrite the optimization problem so that solver outputs the value of $\partial H(x)/\partial x_j$.

Corollary 1. *The optimization problem,*

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

is equivalent to

$$\begin{aligned} H(x) \quad = \quad \max_{a,y} \;& f(y,a) \\ \text{s.t.} \;& g(y,a) = 0, \\ & h(y,a) \geq 0, \\ & x_j - y_j = 0, \quad j = 1, \ldots, d, \end{aligned} \tag{10}$$

implying that

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

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

Corollary 1 takes the optimization problem (9), adds variables y_j, adds constraints $x_j - y_j = 0$, and replaces x by y in the objective function and all other constraints. The Envelope Theorem tells us that $\partial H(x)/\partial x_j$ is the shadow price of the trivial constraint $x_j - y_j = 0$, which can be included in the output of a solver. This frees us from computing the more complex formula (8) in the Envelope Theorem. The extra cost of the reformulated problem is trivial for any modern solver.

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

Algorithm 2. *Hermite Value Function Iteration* (H-VFI)

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

Step *1. Maximization Step. For each $x^i \in \mathbb{X}_t$, $1 \le i \le N_t$, compute*

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

and

$$s_j^i = \tau_j^*(x^i),$$

where $\tau_j^*(x^i)$ is the shadow price of the constraint $x_j^i - y_j = 0$.

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

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

3.2. Hermite Approximation Approaches

Many approximation methods can use Hermite information. Our examples will have smooth value functions, as is often the case in economics problems. In this section we describe polynomial approximation methods that will produce smooth approximations using 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. Chebyshev-Hermite Interpolation

Chebyshev interpolation is often used in L-VFI, and can also be used for H-VFI. If 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 $s^i = V'(x^i)$, then the following system of $2m$ linear equations can produce coefficients for degree $2m-1$ Chebyshev polynomial interpolation on the Hermite data:

$$\begin{cases} \sum_{j=0}^{2m-1} b_j \mathcal{T}_j(z_i) = v_i, & i = 1, \ldots, m, \\ \\ \frac{2}{x_{\max} - x_{\min}} \sum_{j=1}^{2m-1} b_j \mathcal{T}_j'(z_i) = s^i, & i = 1, \ldots, m, \end{cases} \tag{11}$$

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

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, but multidimensional interpolation becomes difficult because the most natural approaches do not have equality between the number of unknown coefficients and the number of items of information. Therefore, we use a least squares approach to fit complete Chebyshev polynomials to Hermite data.

For simplicity, we assume that the approximation range is $[-1,1]^d$ in a d-dimensional approximation problem. It can be easily extended to any range $[x^{\min}, x^{\max}] \subset \mathbb{R}^d$. Assume that we have Hermite data $\{(x^i, v_i, s^i) : i = 1, \ldots, N\}$ on $[-1,1]^d$, where $x^i \in [-1,1]^d$ are d-dimensional approximation nodes, $v_i = V(x^i)$, and $s^i = (s_1^i, \ldots, s_d^i)$ are the gradient of V at x^i, i.e., $s_j^i = \frac{\partial}{\partial x_j} V(x^i)$. The following least square model can produce coefficients for degree n complete Chebyshev polynomial approximation (6) on the Hermite data:

$$\min_{\mathbf{b}} \left\{ \sum_{i=1}^{N} \left(v_i - \sum_{0 \le |\alpha| \le n} b_\alpha \mathcal{T}_\alpha \left(x^i\right) \right)^2 + \sum_{i=1}^{N} \sum_{j=1}^{d} \left(s_j^i - \sum_{0 \le |\alpha| \le n} b_\alpha \frac{\partial}{\partial x_j} \mathcal{T}_\alpha \left(x^i\right) \right)^2 \right\}. \tag{12}$$

Since this is a simple least squares problem without constraints, this is easy and fast to be solved.[1] When the approximation nodes are tensor grids with m nodes in each dimension, there are $N = m^d$ approximation nodes $\{x^i\}$, so there are $(d+1)m^d$ information data $\{v_i, s^i\}$ used for the computing the coefficients of degree n complete Chebyshev polynomial approximation. In this case, it is often good to choose the degree $n = 2m-1$ complete Chebyshev polynomial approximation for the Hermite approximation, as the number of coefficients is $\binom{2m-1+d}{d}$, less than $(d+1)m^d$, for any $d \ge 2$ and $m > 1$.[2]

3.3. Comparison with Previous Work

The key idea in H-VFI is generating and using gradient information. Some of the ideas we outlined above in this paper have been discussed in the lit-

[1]The objective in (12) 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.

[2]This can be proved recursively.

erature, but this is the first complete discussion of the issues. Philbrick and Kitanidis (2001) described the generation and use of Hermite information but did not invoke the Envelope Theorem. Instead they used a formula that sometimes required computing how the optimal choice depends locally on the state, a much more costly and complex procedure. They also used tensor products of splines for the approximation, whereas the multidimensional Hermite approximation approach described in Section 3.2 is more flexible. They report at most three-digit accuracy for their examples of dimension four or less, but less than two digits for higher dimensions. Our examples presented below will achieve higher accuracy. Wang and Judd (2000) combined Hermite approximation with shape preservation, but used a very inefficient approach to multidimensional approximation. Another related method is the shape-preserving rational function spline Hermite interpolation (Cai and Judd, 2012a), but that was only for one-dimensional problems. More important, none of those papers documented the actual performance of H-VFI relative to L-VFI in economically relevant problems. We now turn to how H-VFI performs on substantive economic problems.

4. Application

This section applies H-VFI algorithm to solve multi-stage portfolio optimization problem using 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 present a numerical example of H-VFI to solve multi-stage portfolio optimization problems, and compare it with L-VFI. The dynamic portfolio optimization problem assumes that there are d stocks and one bond available for investment over the stages $t = 0, 1, \ldots, T-1$, and that the investor's objective is to maximize expected utility of wealth at 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 $R = (R_1, \ldots, R_d)^\top$.

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 state variable. In period t, the investor chooses a new portfolio $S_t = (S_{1,t}, \ldots, S_{d,t})^\top$ where $S_{i,t}$ is the market value stock i chosen by the investor in time t. The value of wealth invested in the bond is $B_t = W_t - \mathbf{e}^\top S_t$ where e is the column vector with its i-th element $\mathrm{e}_i = 1$ for all i. Thus, the wealth at time $t+1$ is

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

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

We want to find an optimal portfolio S_t at each time t such that the expected terminal utility is maximized, i.e.,

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

where $u(W) = W^{1-\gamma}/(1-\gamma)$ for some constant $\gamma > 0$ and $\gamma \neq 1$. Moreover, we assume that borrowing or shorting is not allowed in this example, i.e., $B_t \geq 0$ and $S_t \geq 0$ for all t.

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

$$\begin{aligned} V_t(W) \;\; &= \;\; \max_{B,S \geq 0} \;\; \mathbb{E}\{V_{t+1}(R_f B + R^\top S)\}, \\ &\qquad \text{s.t.} \quad W - B - \mathbf{e}^\top S = 0, \end{aligned} \tag{13}$$

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

4.1.1. True Solution

Since the terminal utility function is $u(W) = W^{1-\gamma}/(1-\gamma)$, we know that $V_t(W) = \alpha_t W^{1-\gamma}$, where

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

for any $t < T$, where $\alpha_T = 1/(1-\gamma)$, and x^* is the optimal allocation ratios of money invested in the stocks when the current-stage wealth is 1. This tells us that the optimal solutions are $B_t^* = \left(1 - \mathbf{e}^\top x^*\right) W_t$ and $S_t^* = x^* W_t$ for all time t and any wealth W_t. Moreover, x^* can be easily obtained from computing a single-period optimization problem with the terminal power utility function at $W_0 = 1$. Therefore, we will use this as the true solution and then do our accuracy tests for H-VFI.

4.1.2. Bounded Normal Random Variable

In a dynamic portfolio optimization problem, a typical assumption about the stocks is that they have log-normal returns. This means that the next-stage wealth could be any number in $(0, \infty)$, and then makes it hard to approximate the value function well, particularly when the value function tends to $-\infty$ when wealth goes to 0 and also tends to 0 when wealth goes to infinity, like what we have when the terminal utility function is the power utility with $\gamma > 1$. In this example, we assume that the stocks have a bounded return vector having a distribution close to be log-normal.

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

$$\Psi = \frac{1 - e^{-\kappa\varsigma}}{1 + e^{-\kappa\varsigma}} \Upsilon, \tag{14}$$

where Υ and κ are two 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 of Υ, we would like to choose a corresponding κ so that Ψ has a unit variance. For example, if we set $\Upsilon = 4$, then we will choose $\kappa = 0.532708$. Thus, Ψ is close to ς, particularly in the range $(-2, 2)$ if we choose $\Upsilon \geq 4$. See Cai, Judd and Lontzek (2012c) for more details.

Therefore, in this example we assume that the stocks have a bounded log-normal return $R = (R_1, \ldots, R_d)^\top$, i.e.,

$$\log(R_j) = \mu_i + \frac{1 - e^{-\kappa\varsigma_j}}{1 + e^{-\kappa\varsigma_j}} \Upsilon\sigma_j, \tag{15}$$

where ς_j is a standard normal random variable for $j = 1, \ldots, d$, and the correlation matrix of $(\varsigma_1, \ldots, \varsigma_d)$ is Σ.

4.1.3. Multivariate Numerical Integration

In the objective function of the Bellman equation (13), we need to compute the expectation of V_{t+1}. When we assume that the stocks have a bounded log-normal return vector R, this expectation can be computed by product Gaussian-Hermite quadrature efficiently. From the formula (15), we know that $V_{t+1}(R_f B + R^\top S)$ in the objective function of (13) can be rewritten as $g(\varsigma)$ for some function g where ς is the corresponding standard normal random variable vector with a correlation matrix Σ.

Since Σ must be a positive semi-definite matrix from the covariance property, we can do the Cholesky factorization, i.e., find a lower triangular matrix L such that $\Sigma = LL^\top$. Therefore,

$$\begin{aligned}
& \mathbb{E}\left\{V_{t+1}(R_f B + R^\top S)\right\} = \mathbb{E}\{g(\varsigma)\} \\
= \quad & \left((2\pi)^d \det(\Sigma)\right)^{-1/2} \int_{\mathbb{R}^d} g(x) e^{-x^\top \Sigma^{-1} x/2} dx \\
= \quad & \left((2\pi)^d \det(L)^2\right)^{-1/2} \int_{\mathbb{R}^d} g\left(\sqrt{2}Lx\right) e^{-x^\top x} 2^{d/2} \det(L) dx \\
\doteq \quad & \pi^{-\frac{d}{2}} \sum_{i_1=1}^{N} \cdots \sum_{i_d=1}^{N} \omega_{i_1} \cdots \omega_{i_d} g\Bigg(\sqrt{2}L_{1,1}x_{i_1}, \\
& \sqrt{2}(L_{2,1}x_{i_1} + L_{2,2}x_{i_2}), \cdots, \sqrt{2}(\sum_{j=1}^{d} L_{d,j}x_{i_j})\Bigg),
\end{aligned}$$

where ω_i and x_i 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.

4.1.4. 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 and has a large magnitude at nearly 0 and also is very flat at a large wealth. So it will be hard to approximate the value function well on the state variable W if W has a small lower bound and a large upper bound. Thus, 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, while they are sparser when they are closer to the large upper bound.

To solve this problem, we will use $w = \log(W)$ as our state variable. Experience shows that this approach helps construct accurate polynomial approximations of the value function. If we want to solve the problem for $W \in \left(\underline{W}, \overline{W}\right)$, we approximate the value function with

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

for any $w \in (\underline{w}, \overline{w})$, where $\underline{w} = \log(\underline{W})$ and $\overline{w} = \log(\overline{W})$, and $\mathcal{T}_j$ are the Chebyshev basis polynomials. Moreover, the Chebyshev nodes are $w_i = (z_i + 1)(\overline{w} - \underline{w})/2 + \underline{w}$ with $z_i = -\cos((2i-1)\pi/(2m))$ for $i = 1, \ldots, m$, so we can get the Chebyshev coefficients b_j by using Chebyshev regression algorithm or by solving the linear system (11) for Hermite interpolation.

Since shorting or borrowing is not allowed in our example, and the i-th stock return is assumed to be a bounded log-normal random variable in $\left(e^{\mu_i - \Upsilon\sigma_i}, e^{\mu_i + \Upsilon\sigma_i}\right)$, we can obtain the ranges $[\underline{W}_t, \overline{W}_t]$ 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 $[\underline{W}_0,\ \overline{W}_0]$. We see that the ranges are expanding exponentially across the time. However, the bounds of w_t is

$$\begin{aligned} \underline{w}_{t+1} &= \min_{j=1,\ldots,d} \{\mu_j - \Upsilon\sigma_j\} + \underline{w}_t, \\ \overline{w}_{t+1} &= \max_{j=1,\ldots,d} \{\mu_j + \Upsilon\sigma_j\} + \overline{w}_t, \end{aligned}$$

with $\underline{w}_0 = \log(\underline{W}_0)$ and $\overline{w}_0 = \log\left(\overline{W}_0\right)$, so the ranges are expanding just linearly.

4.1.5. Numerical Result

In this numerical example, we choose $d = 3$ stocks, and they have the bounded log-normal return $R = (R_1, R_2, R_3)^\top$, while the mean of $\log(R)$ is $\mu = (0.04875, 0.04875, 0.04875)^\top$, the standard deviation of $\log(R)$ is $\sigma = (0.15, 0.15, 0.15)^\top$, and the correlation matrix of corresponding $(\zeta_1, \zeta_2, \zeta_3)$ in the formula (15) is

$$\Sigma = \begin{bmatrix} 1 & 0.8 & 0.6 \\ 0.8 & 1 & 0.7 \\ 0.6 & 0.7 & 1 \end{bmatrix}.$$

The interest rate is $r = 0.03$, and the relative risk aversion coefficient is $\gamma = 2$ for the terminal power utility function, and the initial wealth bound is $[\underline{W}_0,\ \overline{W}_0] = [0.9, 1.1]$. We choose $\Upsilon = 4$ and $\kappa = 0.532708$ in the formula (15) so that $\log(R_i)$ is bounded in $(-0.55125, 0.64875)$, i.e., R_i is bounded in $(0.57623, 1.91315)$, which is an appropriate range in reality.

The Envelope Theorem implies $V_t'(W) = \lambda^*(W)$, where $\lambda^*(W)$ is the shadow price for the constraint $W - B - \mathbf{e}^\top S = 0$, so we can get the slope

Table 1: Relative Errors and Running Times of L-VFI or H-VFI for Dynamic Portfolio Optimization

m	L-VFI error	H-VFI error	L-VFI time	H-VFI time
5	0.8	0.00327	9 seconds	10 seconds
10	0.00328	1.3×10^{-5}	12 seconds	17 seconds
20	2.0×10^{-6}		33 seconds	

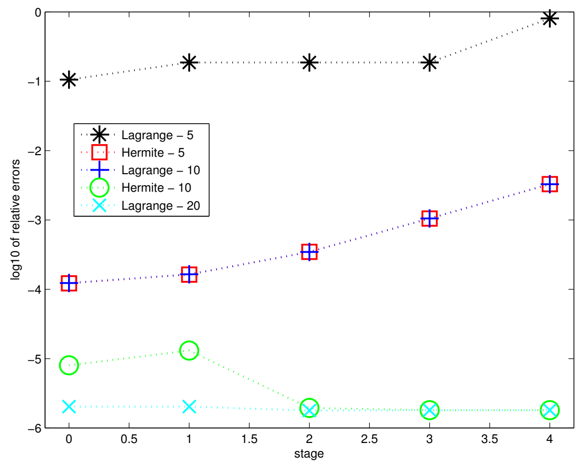
information of the value functions and apply H-VFI without using a dummy variable. The computational results of L-VFI or H-VFI are given by our GAMS (McCarl, 2011) code, and the optimization solver in the Maximization Step of DP is CONOPT. We use the Chebyshev polynomial interpolation as the approximation method, and use the product Gaussian-Hermite quadrature formula with 7 quadrature nodes in each dimension to compute the expectation in the Bellman equation.

Table 1 lists the relative errors of optimal stock allocations and running times[3] of L-VFI or H-VFI, where the column "L-VFI error" lists the $\mathcal{L}^\infty$ relative errors of optimal stock allocations over all stages and all stocks using L-VFI, and the column "L-VFI time" lists the running times of L-VFI, for different number of approximation nodes, m. Similarly, "H-VFI error" and "H-VFI time" list the errors and times of H-VFI. Figure 1 also shows the relative errors of optimal stock allocations for each stage using L-VFI or H-VFI, where the errors are $\mathcal{L}^\infty$ relative errors of optimal stock allocations over all stocks at stage t. The points labeled with "Lagrange - 5" are the errors of L-VFI with 5 approximation nodes, and "Hermite - 5" are for H-VFI with the same 5 approximation nodes. Similarly, "Lagrange - 10" and "Hermite - 10" are for 10 approximation nodes, and "Lagrange - 20' is for 20 nodes.

On the same m Chebyshev nodes, L-VFI uses degree $m-1$ Chebyshev polynomial interpolation on Lagrange data while the Chebyshev coefficients are computed by the regression algorithm; H-VFI uses degree $2m-1$ Chebyshev polynomial interpolation on Hermite data while the Chebyshev coefficients are computed by solving the linear system (11). From Table 1 and Figure1, we see that H-VFI obtains higher accuracy than L-VFI using the same number of approximation nodes, but only takes a bit more computational time which is caused by the almost double degree polynomial approxi-

[3]All the examples are run on a single core of a Mac laptop with a 2.5 GHz processor.

Figure 1: Errors of H-VFI or L-VFI for Dynamic Portfolio Optimization



mation. Moreover, if L-VFI wants to achieve the same accuracy with H-VFI, it should use around $2m$ approximation nodes and the same degree polynomial as H-VFI using m nodes, and it takes nearly double time of H-VFI.

When there one kinks on the value functions, e.g., the terminal utility function is $-(W-K)^{-1}$ for some positive K, H-VFI still has higher accuracy than L-VFI using the same approximation nodes. However, a better usage of slope information for this kind of problems is the shape-preserving rational function spline Hermite interpolation, See Cai and Judd (2012a) for more details.

4.2. Stochastic Optimal Growth Problems

A deterministic optimal growth problem is to find the optimal consumption function and the optimal labor supply function such that the total utility over the T-horizon time is maximal[4], i.e.,

$$\begin{aligned} V_0(k_0) = \max_{k_t,c_t,l_t} \quad & \sum_{t=0}^{T-1} \beta^t u(c_t,l_t) + \beta^T V_T(k_T), \\ \text{s.t.} \quad & k_{t+1} = F(k_t,l_t) - c_t, \quad 0 \le t < T, \end{aligned} \tag{16}$$

[4]See Judd (1998) for a detailed description of this.

where k_t is the capital stock at time t with k_0 given, c_t is the consumption, l_t is the labor supply, β is the discount factor, $F(k,l)$ is the aggregate production function, $V_T(x)$ is a given terminal value function, and $u(c_t,l_t)$ is the utility function.

When the capital stock is dependent on a random economic shock θ, the optimal growth problem (16) becomes a stochastic dynamic optimization problem,

$$\begin{aligned} V_0(k_0,\theta_0) = \max_{k_t,c_t,l_t} \quad & \mathbb{E}\left\{\sum_{t=0}^{T-1}\beta^t u(c_t,l_t) + \beta^T V_T(k_T,\theta_T)\right\}, \\ \text{s.t.} \quad & k_{t+1} = F(k_t,l_t,\theta_t) - c_t, \quad 0 \le t < T, \\ & \theta_{t+1} = h(\theta_t,\epsilon_t), \quad 0 \le t < T, \end{aligned} \tag{17}$$

where θ_t is a discrete time process with its transition function h, ϵ_t is a serially uncorrelated random process, and $F(k,l,\theta)$ is the aggregate production function dependent on the economic shock. This objective function is time-separable, so this can be modeled as a DP problem (2), and then we can use DP methods to solve it.

In the examples, the discount factor is $\beta = 0.95$, the aggregate production function is $F(k,l,\theta) = k + \theta A k^{\psi} l^{1-\psi}$ with $\psi = 0.25$ and $A = (1-\beta)/(\psi\beta)$, and the utility function is the same with (18). The terminal value function is $V_T(k,\theta) = u(F(k,1,1)-k,1)/(1-\beta)$. The range of capital stocks for approximation is $[0.2, 3]$. We assume that θ_t is a Markov chain, the possible values of θ_t are $\vartheta_1 = 0.9$ and $\vartheta_2 = 1.1$, and the probability transition matrix from θ_t to θ_{t+1} is

$$\left[\begin{array}{cc} 0.75 & 0.25 \\ 0.25 & 0.75 \end{array}\right],$$

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

The utility function is

$$u(c,l) = \frac{(c/A)^{1-\gamma} - 1}{1-\gamma} - (1-\psi)\frac{l^{1+\eta} - 1}{1+\eta}. \tag{18}$$

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

4.2.1. DP Solution of Stochastic Optimal Growth Problems

The DP formulation of the stochastic optimal growth problem (17) in stochastic extension of Algorithm 2 is:

$$\begin{aligned} V_t(k,\theta) &= \max_{k^+,c,l,y} \quad u(c,l)+\beta\mathbb{E}\left\{V_{t+1}(k^+,\theta^+)\right\}, \\ & \quad \text{s.t.} \quad k^+ = F(y,l,\theta)-c, \\ & \qquad\quad\ \theta^+ = h(\theta,\epsilon), \\ & \qquad\quad\ k-y=0, \end{aligned} \tag{19}$$

for $t < T$, where k is the continuous state, θ is the discrete state, k^+ and θ^+ are next-stage continuous and discrete states respectively, (c,l) are the control variables, ϵ is a random variable, and the terminal value function $V_T(k,\theta)$ is given. Here the dummy variable y and the trivial constraint $k-y=0$ are used in order to get the slopes of value functions directly from the optimization solver as described in H-VFI.

4.2.2. Tree Method

To solve the stochastic model (17) using nonlinear programming, we can apply a tree method that is a generalized approach from solving the deterministic model (16) by nonlinear programming. Assume that θ_t is a Markov chain with possible states, $\vartheta_1,\ldots,\vartheta_m$, and the probability of going from state ϑ_i to state ϑ_j in one step is

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

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

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

for $1 \le n \le m^t$, where $\mathrm{mod}(n-1,m)$ is the remainder of division of $(n-1)$ by m, and $\lfloor (n-1)/m \rfloor$ is the integer part of $(n-1)/m$.

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

$$\max_{c_{t,n},l_{t,n}} \quad \sum_{t=0}^{T-1} \beta^t \sum_{n=1}^{m^t} (P_{t,n} u(c_{t,n}, l_{t,n})) + \beta^T \sum_{n=1}^{m^{T-1}} P_{T-1,n} \sum_{j=1}^{m} q_{\mathrm{mod}(n-1,m)+1,j} V_T(k_{T,n}, \vartheta_j), \tag{20}$$

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

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

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

This approach is practical for only small values of m and T. In our examples, we set we have $m = 2$ 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 error analysis of numerical DP algorithms. One can look at problems with thousands of paths, but the tree method becomes infeasible as we increase m and T; see Cai (2009).

4.2.3. Error Analysis

We examine the errors for the stochastic model in the same manner we did for the deterministic optimal growth problems: We apply nonlinear programming to get the "true" optimal solution on the model (20) for every test point of initial capital $k_0 \in [\underline{k}, \bar{k}]$ and every possible initial discrete state θ_0, and then use them to check the accuracy of the computed optimal solution from L-VFI or H-VFI on the model (19).

Table 2 lists errors of optimal solutions computed by L-VFI or H-VFI when $T = 5$. The computational results are given by our GAMS code, where we use degree $2m-1$ or $m-1$, respectively, Chebyshev polynomial interpolation on m expanded Chebyshev nodes. And in the Maximization Step of DP, we use SNOPT[5] (Gill et al., 2005).

[5]The optimality and feasibility tolerances are set to be 10^{-9}.

Table 2: Errors of optimal solutions of L-VFI or H-VFI for stochastic growth problems

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

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

The errors for optimal consumptions at time 0 are computed by

$$\max_{k\in[0.2,3],\theta\in\{0.9,1.1\}} \frac{|c_{0,\mathrm{DP}}^*(k,\theta)-c_0^*(k,\theta)|}{|c_0^*(k,\theta)|},$$

where $c_{0,\mathrm{DP}}^*$ is the optimal consumption at time 0 computed by L-VFI or H-VFI on the model (19), and c_0^* is the "true" optimal consumption computed by nonlinear programming on the model (20). The similar formula applies to compute errors for optimal labor supply at time 0.

Table 2 shows that H-VFI is more accurate than L-VFI, with about one or two digits accuracy improvement. For example, row one in Table 2 assumes $\gamma = 0.5$, $\eta = 0.1$, and $m = 5$ approximation nodes. For this case, the error in consumption is 0.11 for L-VFI, and drops to 0.013 when we use H-VFI. Similarly the error in labor supply is 0.19 for L-VFI, and drops to 0.018 when we use H-VFI. The rest of Table 2 examines different γ and η, and shows the same patterns for the reduction of errors when we switch from L-VFI to H-VFI. For both consumption and labor supply, H-VFI is about 10 times more accurate than L-VFI using 5 approximation nodes. Row two in Table 2 chooses $m = 10$ approximation nodes, and in this case both consumption and labor supply errors drop by a factor about 200 when we switch from L-VFI to H-VFI. For these examples, both DP algorithms are quite fast, so we do not list their running times for comparison. However, we will show them in the next multidimensional examples.

4.3. Three-Country Optimal Growth Problems

The more important usage of slope information are in multidimensional DP problems, as it will take much more time to compute the optimal policy for one approximation node. But for a d-dimensional DP problem, H-VFI uses not only m values but also $d \times m$ slopes which comes almost freely at cost, while L-VFI can only use m values. Thus, H-VFI will reduce the computational time significantly than L-VFI, if one wants to achieve the same accuracy of optimal solution. We show this by solving multi-country optimal growth problems.

We assume that there are d countries, and let $k_t = (k_{t,1}, \dots, k_{t,d})$ denote the capital stocks of these countries which is a d-dimensional continuous state vector at time t. Let $l_t = (l_{t,1}, \dots, l_{t,d})$ denote elastic labor supply levels of the countries which is a d-dimensional continuous control vector variable at time t. Assume that the net production of country j at time t is

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

with $A = (1-\beta)/(\psi\beta)$, for $j = 1, \ldots, d$. Let $c_t = (c_{t,1}, \ldots, c_{t,d})$ denote consumption of the countries which is another d-dimensional continuous control vector variable at time t. The utility function is

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

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

$$\begin{aligned} V_0(k_0) \quad = \quad \max_{k_t, I_t, c_t, l_t} \quad & \sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T V_T(k_T), \qquad (21) \\ \text{s.t.} \quad & k_{t+1,j} = (1-\delta)k_{t,j} + I_{t,j}, \quad j = 1, \ldots, d, \\ & \Gamma_{t,j} = \frac{\zeta}{2} k_{t,j} \left(\frac{I_{t,j}}{k_{t,j}} - \delta \right)^2, \quad j = 1, \ldots, d, \\ & \sum_{j=1}^{d} (c_{t,j} + I_{t,j} - \delta k_{t,j}) = \sum_{j=1}^{d} \left(f(k_{t,j}, l_{t,j}) - \Gamma_{t,j} \right), \end{aligned}$$

where δ is the depreciation rate of capital, $I_{t,j}$ is the investment of country j, $\Gamma_{t,j}$ is the investment adjustment cost of country j, and ζ governs the intensity of the friction. The functional forms for utility and production imply that the steady state of the infinite horizon deterministic optimal growth problems is $k_{ss,j} = 1$, and the optimal consumption, labor supply and investment at the steady state are respectively $c_{ss,j} = A$, $l_{ss,j} = 1$, and $I_{ss,j} = \delta$, for $j = 1, \ldots, d$. Detailed discussion of multi-country growth models with infinite horizon can be seen in Den Haan et al (2011), Juillard and Villemot (2011), and a nonlinear programming method for dynamic programming is introduced in Cai et al. (2012d) to solve the multi-country growth model with infinite horizon.

The DP formulation of the multi-country model (21) in H-VFI is

$$\begin{aligned} V_t(k) &= \max_{k^+,I,c,l,y} \quad u(c,l) + \beta V_{t+1}(k^+), \qquad (22) \\ &\quad \text{s.t.} \quad k_j^+ = (1-\delta)y_j + I_j, \quad j = 1,\ldots,d, \\ &\quad\quad \Gamma_j = \frac{\zeta}{2} y_j \left(\frac{I_j}{y_j} - \delta\right)^2, \quad j = 1,\ldots,d, \\ &\quad\quad \sum_{j=1}^{d} (c_j + I_j - \delta y_j) = \sum_{j=1}^{d} (f(y_j, l_j) - \Gamma_j), \\ &\quad\quad k_j - y_j = 0, \quad j = 1,\ldots,d, \end{aligned}$$

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 value functions at k directly from the optimization solver as described in Algorithm 2, for $j = 1, \ldots, d$.

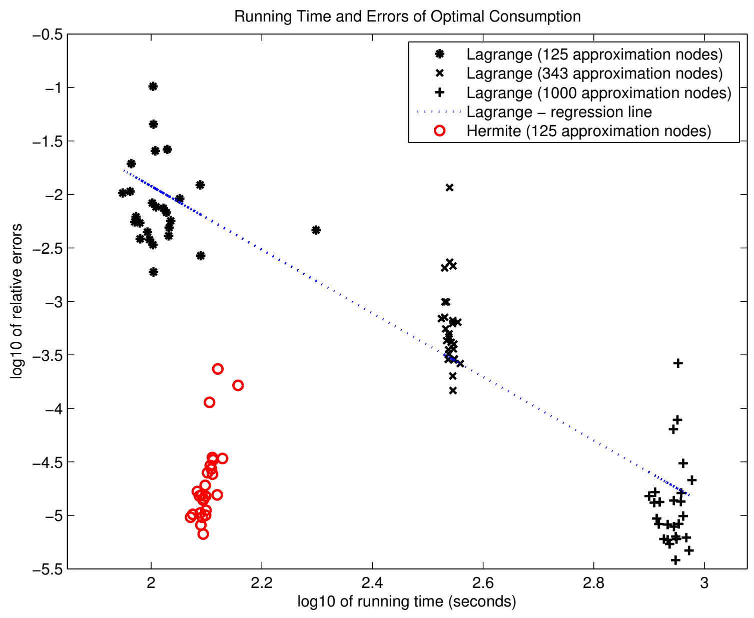
In this subsection, we let $d = 3$. So this is a DP example with 3-dimensional continuous states. Let $T = 5$, $\psi = 0.36$, $\delta = 0.025$, $\zeta = 0.5$, and let the capitals range be $[0.5, 1.5]^3$. The terminal value function is $V_T(k) = u(f(k,1),1)/(1-\beta)$. We get the true solution of this three-country model by solving the model (21) directly using CONOPT in GAMS.

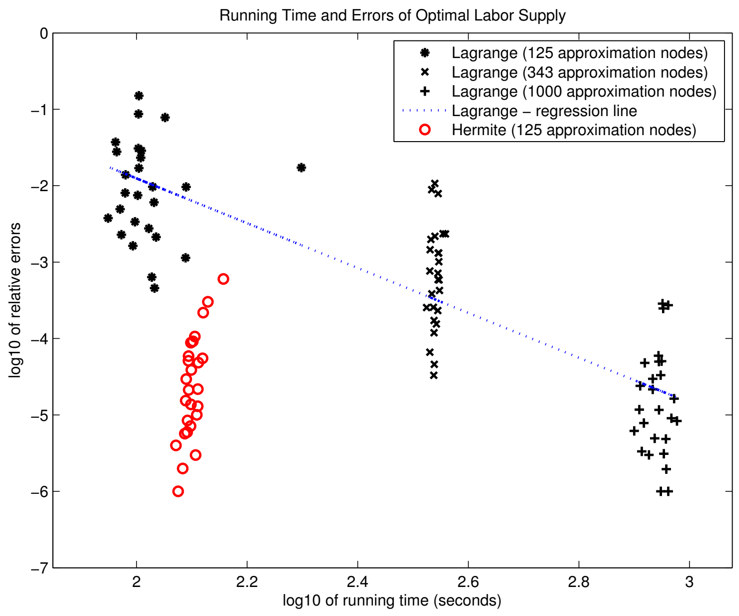
Figure 2 displays the relative errors of optimal consumption, optimal labor supply, and running times[6] of L-VFI or H-VFI using tensor grid of $m = 5, 7, 10$ expanded Chebyshev nodes in each dimension, for various $\beta = 0.9, 0.95, 0.99$, $\gamma = 0.5, 2, 5$ and $\eta = 0.2, 1, 5$. For the Lagrange approximation using m nodes in each dimension, we use degree $m-1$ complete Chebyshev polynomials and apply the regression algorithm to compute the coefficients. And for the Hermite approximation using m nodes in each dimension, we use degree $2m-1$ complete Chebyshev polynomials and apply the least square method (12) to compute the coefficients. All the codes are written in GAMS, and we use CONOPT as the optimization solver in the Maximization Step of L-VFI and H-VFI. 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, and the circles represent Hermite approximation with $m = 5$, for various β, γ, η.

From comparing the circles and the stars in Figure 2, we see that, for any

[6]All the examples are run on one core of a single core of a Mac laptop with a 2.5 GHz processor.

Figure 2: L-VFI vs H-VFI for Three-Country Optimal Growth Problems





combination of β, γ, η, H-VFI achieves higher accuracy than L-VFI using the same number of approximation nodes, with about three digits higher accuracy when $m = 5$. Moreover, the running times of H-VFI are only around 120 seconds, just a bit more than L-VFI (around 100 seconds).

From comparing the circles and the marks in Figure 2, we see that, H-VFI with $m = 5$ approximation nodes in each dimension are not only nearly 3 times faster but also about one more digit higher accuracy than Lagrange approximation with $m = 7$.

From comparing the circles and the pluses in Figure 2, we see that if L-VFI wants to have the same accuracy as Hermite approximation with $m = 5$, it needs nearly 8 times of approximation nodes ($m = 10$) and also nearly 8 times of computational time than H-VFI, for the three-country optimal growth problems.

4.4. Six-Country Optimal Stochastic Growth Problems

Our last example shows that the advantage of H-VFI will be even greater for higher dimensional problems. We use the multi-country model (21) with $d = 6$ countries and the similar setting as the previous examples, but we assume that the net productions of all countries are dependent on a random economic shock θ_t, which is a Markov chain. That is, the net production of country j at time t is

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

where 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

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

Table 3: H-VFI vs L-VFI for Six-Dimensional Stochastic Problems

	error of c_0^*		error of l_0^*		running times	
m	L-VFI	H-VFI	L-VFI	H-VFI	L-VFI	H-VFI
3	$3.8(-2)$	$3.6(-3)$	$5.4(-2)$	$5.2(-3)$	0.3 hour	0.67 hour
5	$5.5(-3)$		$8.2(-3)$		8.74 hours	
6	$3.1(-3)$		$4.5(-3)$		36.6 hours	

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

for $t = 0, \ldots, T-1$. Therefore, we have the stochastic version of the multi-country model (21):

$$\begin{aligned} V_0(k_0, \theta_0) \quad = \quad & \max_{k_t, I_t, c_t, l_t} \quad \mathbb{E}\left\{\sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T V_T(k_T, \theta_T)\right\}, \qquad (23) \\ & \text{s.t.} \quad k_{t+1,j} = (1-\delta)k_{t,j} + I_{t,j}, \quad j = 1, \ldots, d, \\ & \qquad \Gamma_{t,j} = \frac{\zeta}{2} k_{t,j} \left(\frac{I_{t,j}}{k_{t,j}} - \delta\right)^2, \quad j = 1, \ldots, d, \\ & \qquad \sum_{j=1}^{d} (c_{t,j} + I_{t,j} - \delta k_{t,j}) = \sum_{j=1}^{d} \left(f(k_{t,j}, l_{t,j}, \theta_t) - \Gamma_{t,j}\right), \\ & \qquad \theta_{t+1} = g(\theta_t, \epsilon_t). \end{aligned}$$

Then we can also solve this problem with L-VFI or H-VFI.

In this example, we choose $\beta = 0.95$, $\gamma = 2$ and $\eta = 1$. The number of decision stages is $T = 5$, and the capitals range is $[0.5, 1.5]^6$. We apply the tree model (20) and use CONOPT in GAMS code to compute the "true" solutions at test points of κ_0 and each possible value of θ_0.

Table 3 lists the running times[7], and relative errors of optimal consumption and labor supply of L-VFI and H-VFI using tensor grid of $m = 3, 5, 6$ expanded Chebyshev nodes in each dimension on $[0.5, 1.5]^6$. On the tensor grid of m nodes in each dimension, Lagrange approximation uses degree $m-1$ complete Chebyshev polynomials, and Hermite approximation uses degree $2m-1$ complete Chebyshev polynomials and the least squares model (12) is applied to obtain coefficients.

[7]All the examples are run on one core of a single core of a Mac laptop with a 2.5 GHz processor.

From Table 3, we see that H-VFI using $m = 3$ approximation nodes in each dimension takes only 0.67 hours to achieves the higher accuracy than L-VFI using $m = 5$ approximation nodes in each dimension which takes 8.74 hours, about 13 times computational time of H-VFI. Moreover, if L-VFI wants to use $m = 6$ approximation nodes in each dimension to achieve a bit higher accuracy than H-VFI, the running time is up to 36.6 hours, about 55 times computational time of H-VFI.[8]

5. Conclusion

We have shown that gradient of the value function can be obtained easily and almost freely during value function iteration, and that using it in Hermite approximation approximation methods will produce significant improvement in the efficiency of numerical dynamic programming. These conclusions are supported by examples from dynamic portfolio optimization problems and multi-dimensional optimal growth problems. We show that this improvement can be used to either improve accuracy for a given amount of computation time, or to attain the same accuracy much faster than with L-VFI. These examples also indicate that H-VFI will be particularly valuable in solving multidimensional dynamic programming problems.

[8]All the running times can be dramatically reduced if we use Fortran code and NPSOL solver (Gill et al., 1994), but the relative difference between Hermite and Lagrange approximation is still almost the same with GAMS. Moreover, for higher dimensional problems, we can solve them using Fortran code in a reasonable time. We conclude that the speed advantage of H-VFI over L-VFI will increase with dimension.

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

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

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

[4] Cai, Y., and K.L. Judd (2012a). Dynamic programming with shape-preserving rational spline Hermite interpolation. *Economics Letters*, Vol. 117, No. 1, 161–164.

[5] Cai, Y., and K.L. Judd (2012b). Shape-preserving dynamic programming. *Mathematical Methods of Operations Research*, DOI: 10.1007/s00186-012-0406-5.

[6] Cai, Y., K.L. Judd and T.S. Lontzek (2012c). DSICE: A Dynamic Stochastic Integrated Model of Climate and Economy. RDCEP working paper No. 12-02.

[7] Cai, Y., K.L. Judd, T.S. Lontzek, V. Michelangeli, and C.-L. Su (2012d). Nonlinear Programming method for dynamic programming. Hoover working paper.

[8] Den Haan, W.J., K.L. Judd and M. Juillard (2011). Computational suite of models with heterogeneous agents II: Multicountry real business cycle models. Journal of Economic Dynamics & Control, 35, 175–177.

[9] Gill, P., W. Murray, M.A. Saunders and M.H. Wright (1994). User's Guide for NPSOL 5.0: a Fortran Package for Nonlinear Programming. Technical report, SOL, Stanford University.

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

[11] Judd, K.L. (1998). *Numerical Methods in Economics.* The MIT Press.

[12] Juillard, M., and S. Villemot (2011). Multi-country real business cycle models: Accuracy tests and test bench. *Journal of Economic Dynamics & Control*, 35, 178–185.

[13] McCarl, B., et al. (2011). McCarl Expanded GAMS user guide version 23.6. http://www.gams.com/mccarl/mccarlhtml/. Accessed 06 September 2012.

[14] Philbrick, C.R., and P.K. Kitanidis (2001). Improved Dynamic Programming Methods for Optimal Control of Lumped Parameter Stochastic Systems. *Operations Research*, 49(3), 398–412.

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

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

[17] Wang, S.P., and K.L. Judd (2000). Solving a savings allocation problem by numerical dynamic programming with shape-preserving interpolation. *Computers and Operations Research*, Volume 27, Issue 5, 399–408.