CHAPTER EIGHT

Advances in Numerical Dynamic Programming and New Applications

Yongyang Cai and Kenneth L. Judd
Hoover Institution & NBER, USA

Contents

Handbook of Computational Economics, Volume 3
ISSN 1574-0021, http://dx.doi.org/10.1016/B978-0-444-52980-0.00008-6

1. INTRODUCTION

All dynamic economic problems are multistage decision problems, and their nonlinearities make them numerically challenging. Dynamic programming is the standard approach for any time-separable problem. If state variables and control variables are continuous, and the problem is a concave maximization problem, then its value function is continuous, concave, and often differentiable. Any numerical procedure needs to approximate the value function, but any such approximation will be imperfect since computers cannot model the entire space of continuous functions. Many dynamic programming problems are solved by value function iteration, where the period t value function is computed from the period $t+1$ value function, and the value function at the terminal time T is known.

Dynamic programming problems can generally be formulated by the following Bellman equation (Bellman, 1957):

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

where x is the vector of continuous state variables in $\mathbb{R}^d$, and θ is an element of the set of discrete state vectors, $\Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^{d'}$ (where D is the number of different discrete state vectors in $\mathbb{R}^{d'}$). $V_t(x,\theta)$ is the value function at time $t \leq T$, and the terminal value function, $V_T(x,\theta)$, is given. The decision maker chooses a vector of action variables, a, where the choice is constrained by $a \in \mathcal{D}(x,\theta,t)$. We let x^+ denote the value of the continuous state variables in the next period, and assume that the law of motion is a time-specific function g_t at time t. Similarly, θ^+ is the next-stage discrete state vector in Θ with a transition function h_t at time t. The state transitions may be affected by random shocks, ω and ϵ. At time t, the payoff flow is $u_t(x,a)$, and the overall objective is to maximize the expected discounted sum of utility, using β as the discount factor. We let $\mathbb{E}\{\cdot\}$ denote the expectation operator.

To explain computational ideas that arise often in applications of dynamic programming in economics, we will often use the simple case with no discrete states and no random shocks, assumptions that simplify the Bellman equation (1) to

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

where Γ is the Bellman operator that maps the period $t + 1$ value function V_{t+1} to the period t value function V_t. The Bellman operator is possibly different at each time t, and should be denoted Γ_t.

This chapter focuses on the computational challenges of solving dynamic programming problems. We first address the concerns over the "curse of dimensionality" often raised in theoretical discussions of the computational complexity of solving dynamic programming problems. If the curse of dimensionality were present in nearly all dynamic programming problems then there would be little point in even attempting to solve multidimensional problems. We review the results in the complexity literature but point out the features of a dynamic programming problem that may keep it from being a victim of the curse of dimensionality. Sections 3 and 4 review the range of methods from numerical analysis that we can use to construct efficient algorithms. Section 5 discusses the importance of shape-preservation for constructing stable numerical implementations of value function iteration. Section 6 discusses the use of massive parallelism to solve large dynamic programming problems.

The combination of shape-preserving approximation methods and massive parallelism allows us to solve problems previously considered too challenging for dynamic programming. These ideas are being incorporated in work that extends the range of problems analyzed by stochastic dynamic programming models. Section 7 discusses recent work on portfolio decision making when there are transaction costs. The multidimensional finite-horizon analysis of Section 7 shows us that we can now analyze life-cycle problems far more realistically than is currently the practice in quantitative economic analysis. Section 8 presents a stochastic dynamic general equilibrium extension of DICE, a basic model of interactions between climate and the economy. Previous analyses have been limited by computational tools to examine only deterministic models of global climate change. Section 8 shows that we can now analyze models that come far closer to analyzing risks and uncertainty that are inherent in any discussion of climate change policy.

2. THEORETICAL CHALLENGES

There are two challenges in solving difficult dynamic programming problems. First, numerical methods do not necessarily inherit the contraction properties of the Bellman operator. This creates stability problems that become increasingly challenging as one increases dimension. Second, dynamic programming is often said to suffer from the "curse of dimensionality"; that is, the cost of solving a dynamic programming problem may grow exponentially as the dimension increases. In this section we describe recent theoretical work on these issues.

Rust (1997) and Rust et al. (2002) are two recent papers that prove that the curse of dimensionality is a problem for large classes of dynamic programming problems. However, before one becomes too pessimistic about solving high-dimensional dynamic programming problems, he should remember how the curse of dimensionality is defined. First, it is always a statement about a set of dynamic programming problems, and, second, it says that there is a sequence of problems in that set where the cost explodes exponentially as the dimension rises. The underlying approach is the worst-case analysis. More precisely, it means that for any algorithm, there is a sequence of dynamic programming problems of increasing dimension such that the cost rises exponentially in the dimension. Even if there is only one such example, we still say that there is a curse of dimensionality.

This need not be a major concern. A proof of exponential complexity says nothing about the average cost of using an algorithm to solve a problem. One way to proceed is to find algorithms that grow polynomially on average as dimension increases. This would be a difficult direction, requiring the development of deep mathematical analysis. The other, and more practical, way to proceed is to find formulations of economics problems that avoid the curse of dimensionality. Complexity theory provides guidance on that issue. While the literature is large, a very general treatment is in Griebel and Wozniakowski (2006) which shows that, as long as an unknown function has sufficient smoothness, then there is no curse of dimensionality in computing its derivatives or in approximating it from a finite sample of its values. Therefore, problems with smooth payoffs, smooth transitions, and smooth value functions can avoid the curse of dimensionality. Many problems in economics have no difficulty in satisfying these requirements.

A second problem that arises in numerical solutions is that numerical value function iteration may not be stable. To understand this issue, we need to recall the key property of the Bellman operator. Assume that the Bellman operator Γ maps a bounded value function V to a bounded function, where the state space of V is compact. The critical feature of value function iteration is that Γ is a contraction in $\mathcal{L}^{\infty}$, i.e., $\left\|\Gamma(f)-\Gamma(g)\right\|_{\infty} \leq \beta\left\|f-g\right\|_{\infty}$, for any continuous and bounded functions f and g on the compact state space, if $\beta \in (0,1)$. Numerical methods cannot represent V perfectly. Let L denote the method used to approximate $\Gamma(V)$, implying that the approximation of $\Gamma(V)$ is denoted by $\hat{\Gamma} := L \circ \Gamma$. Various errors in approximation and computing expectations can prevent $\hat{\Gamma}$ from being a contraction even though Γ is. This can lead to nonconvergence or even divergence for numerical value function iteration based on $\hat{\Gamma}$. Stachurski (2008) discusses approximation structures in dynamic programming problems and their impact on the stability of value function iteration. Stachurski shows that if L is nonexpansive, i.e., $\left\|L(f)-L(g)\right\| \leq \left\|f-g\right\|$, then the operator $\hat{\Gamma}$ is also a contraction mapping. He exploits the contractiveness of $\hat{\Gamma}$ to obtain error bounds for the approximate value functions for general nonexpansive approximation methods.

Even though Stachurski discusses stationary infinite-horizon problems, these considerations are equally important in finite-horizon dynamic programming, which is the focus of this chapter. Even if the Bellman operator Γ_t is different at each time t, it is still a contraction operator on its domain. We still want each approximate Bellman operator $\hat{\Gamma}_t$ to have that same property. If, instead, the approximation method implies a possibly expansive operator L, then successive applications of the $\hat{\Gamma}_t$ operators may generate spurious numerical errors and prevent accurate approximations of the value and policy functions. Therefore, the expansiveness considerations in Stachurski (2008) apply to stability issues in finite-horizon dynamic programming.

Nonexpansiveness is related to the concept of shape-preserving approximation. Judd and Solnick (1994) highlighted the computational advantages of such approximations, where the "shapes" of greatest interest are monotonicity and convexity (or concavity). Piecewise linear interpolation is an example of an approximation method which is both nonexpansive and shape-preserving in one dimension. Stachurski (2008) points out that some shape-preserving quasi-interpolants are also nonexpansive.

3. NUMERICAL METHODS FOR DYNAMIC PROGRAMMING

If state and control variables in a dynamic programming problem are continuous, then the value function is a function in $\mathbb{R}^d$, and 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 choose 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, a numerical method focuses on finding the vector of parameters, $\mathbf{b}$, such that $\hat{V}(x; \mathbf{b})$ approximately satisfies the Bellman equation for all times t.

3.1 Outline of the Basic Value Function Iteration Algorithm

Algorithm 1 presents the traditional value function iteration for solving the simple Bellman equation (2). In Algorithm 1, a numerical solution needs only to approximate the value function and solve the optimization problem at a finite number of values for the state variable.

Algorithm 1. Value Function Iteration for the Simple Dynamic Programming Model (2)

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*

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

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.*

The more general case of stochastic dynamic programming and discrete state variables is presented in Algorithm 2. The presence of stochastic disturbances implies the need to compute the expected value function at the next period, which presents a new computational challenge. The presence of discrete states does not create new computational challenges because the representation of the value function is to create an approximation over the continuous states x for each distinct discrete state. In particular, discrete states do not increase the number of dimensions of the continuous portions of the value function.

Algorithm 2. Value Function Iteration for the General Dynamic Programming Model (1)

Initialization. *Given a finite set of* $\theta \in \Theta = \{\theta^j : 1 \le j \le D\} \subset \mathbb{R}^{d'}$ *and the probability transition matrix* $P = \left(p_{j,j'}\right)_{D\times D}$ *where* $p_{j,j'}$ *is the transition probability from* $\theta^j \in \Theta$ *to* $\theta^{j'} \in \Theta$ *for* $1 \le j, j' \le D$, *choose a functional form for* $\hat{V}(x, \theta; \mathbf{b})$ *for all* $\theta \in \Theta$, *and choose the approximation nodes,* $\mathbb{X}_t = \{x_t^i : 1 \le i \le N_t\} \subset \mathbb{R}^d$. *Let* $\hat{V}(x, \theta; \mathbf{b}^T) = V_T(x, \theta)$. *Then for* $t = T-1, T-2, \ldots, 0$, *iterate through steps 1 and 2.*

Step 1. *Maximization step. Compute*

$$v_{i,j} = \max_{a\in\mathcal{D}(x^i,\theta^j,t)} u_t(x^i, \theta^j, a) + \beta\mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\}, \tag{3}$$

for each $x^i \in \mathbb{X}_t$ *and* $\theta^j \in \Theta, 1 \le i \le N_t, 1 \le j \le D$, *where the next-stage discrete state* θ^+ *is random with probability mass function* $\Pr(\theta^+ = \theta^{j'} \mid \theta^j) = p_{j,j'}$ *for each* $\theta^{j'} \in \Theta$, *and* x^+ *is the next-stage state transition from* x^i *and may be also random.*

Step 2. *Fitting step. Using an appropriate approximation method, for each* $1 \le j \le D$, *compute* $\mathbf{b}_j^t$, *such that* $\hat{V}(x, \theta_j; \mathbf{b}_j^t)$ *approximates* $\{(x^i, v_{i,j}) : 1 \le i \le N_t\}$ *data, i.e.,* $v_{i,j} \approx \hat{V}(x^i, \theta^j; \mathbf{b}_j^t)$ *for all* $x^i \in \mathbb{X}_t$. *Let* $\mathbf{b}^t = \left\{\mathbf{b}_j^t : 1 \le j \le D\right\}$.

Algorithm 2 includes three types of numerical problems. First, we need to solve a maximization problem at each node $x^i \in \mathbb{X}_t$ and $\theta^j \in \Theta$. Second, the evaluation of the objective requires us to compute an expectation. Third, we need to efficiently take the data and compute the best fit for the new value function. The challenge is not only to use good numerical methods for each of these steps but also to choose methods that are compatible with each other and jointly lead to efficient algorithms. The next section describes these choices in more detail. More detailed discussion can be found in Cai (2010), Judd (1998), and Rust (2008).

3.2 Typical Applications

Dynamic programming has been applied to numerous economic problems. For the purposes of this chapter, we use two basic applications familiar to readers. These examples will allow us to later illustrate numerical methods in a clear manner.

3.2.1 Optimal Growth Example

We first illustrate our methods with a discrete-time optimal growth problem with one good and one capital stock.[1] The objective is to find the optimal consumption function and the optimal labor supply function such that the total utility over the T-horizon time is maximal, i.e.,

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

where k_t is the capital stock at time t with k_0 given, c_t is the consumption of the good, l_t is the labor supply, $\underline{k}$ and $\bar{k}$ are given lower and upper bound of k_t, β is the discount factor, $F(k, l) = k + f(k, l)$ with $f(k_t, l_t)$ the aggregate net production function, $V_T(x)$ is a given terminal value function, and $u(c_t, l_t)$ is the utility function, and ϵ is a small positive number to avoid the nonpositive consumption or labor supply.

The dynamic programming version of the discrete-time optimal growth problem is the Bellman equation:

$$\begin{aligned} V_t(k) = \max_{c,l} \quad & u(c, l) + \beta V_{t+1}(k^+), \\ \text{s.t.} \quad & k^+ = F(k, l) - c, \\ & \underline{k} \le k^+ \le \bar{k}, \quad c, l \ge \epsilon, \end{aligned} \tag{5}$$

for $t < T$, where $V_T(x)$ is the previously given terminal value function. Here k is the state variable and (c, l) are the control variables.

Using dynamic programming does not make more traditional methods obsolete; in fact, careful applications of dynamic programming will use traditional methods to check solutions. For the finite-horizon optimal growth problem (4), when T is small, we can use a good large-scale optimization package to solve the problem directly, and its solution could be better than the solution of the dynamic programming model (5) given by numerical dynamic programming algorithms because of the numerical approximation errors. Numerical dynamic programming is a problem in infinite-dimensional function spaces and we do not know a priori how flexible our finite-dimensional approximations need to be. Comparing our dynamic programming solution to the solutions from conventional optimization methods can help us determine the amount of flexibility we need to solve for the value function.

When we turn to stochastic versions of the growth model, dynamic programming must be used since conventional optimization methods can no longer be used when either the

[1] Please see Judd (1998) for a detailed description of this.

horizon or number of random states is large. However, as long as the complexity of the value function is only moderately affected by the stochastic terms, the information obtained from conventional methods applied to the deterministic problem will tell us much about the value function for the stochastic problem.

3.2.2 Multistage Portfolio Optimization Example

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

$$W_{t+1} = R_f(W_t - e^\top S_t) + R^\top S_t, \tag{6}$$

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

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

$$V_0(W_0) = \max_{S_t, 0 \le t < T} \mathbb{E}\{u(W_T)\}, \tag{7}$$

where W_T is the terminal wealth derived from the recursive formula (6) with a given W_0, and u is the terminal utility function, and $\mathbb{E}\{\cdot\}$ is the expectation operator.

The dynamic programming model of this multistage portfolio optimization problem is

$$\begin{aligned} V_t(W) = \max_{B,S} \quad & \mathbb{E}\{V_{t+1}(R_f B + R^\top S)\}, \\ \text{s.t.} \quad & B + e^\top S = W, \end{aligned} \tag{8}$$

for $t = 0, 1, \ldots, T-1$, where W is the state variable and S is the control variable vector, and the terminal value function is $V_T(W) = u(W)$. We should add $B \ge 0$ and $S \ge 0$ as bound constraints in the above dynamic programming model, if neither shorting stock nor borrowing bond is allowed.

For small portfolio problems, conventional methods can be used. In the portfolio optimization problem (7), if we discretize the random returns of k stocks as $R = R^{(j)} = (R_{1,j}, \ldots, R_{k,j})$ with probability q_j for $1 \le j \le m$, then it becomes a tree. Figure 1 shows one simple tree with $m = 2$ and $T = 2$ for a portfolio with one bond and one stock. The stock's random return has a probability q_1 to have a return $R_{1,1}$, and the probability $q_2 = 1 - q_1$ to have a return $R_{1,2}$. So there are two scenarios at time 1: $(W_{1,1}, P_{1,1})$ and $(W_{1,2}, P_{1,2})$, and four scenarios at time 2: $(W_{2,1}, P_{2,1}), \ldots, (W_{2,4}, P_{2,4})$.

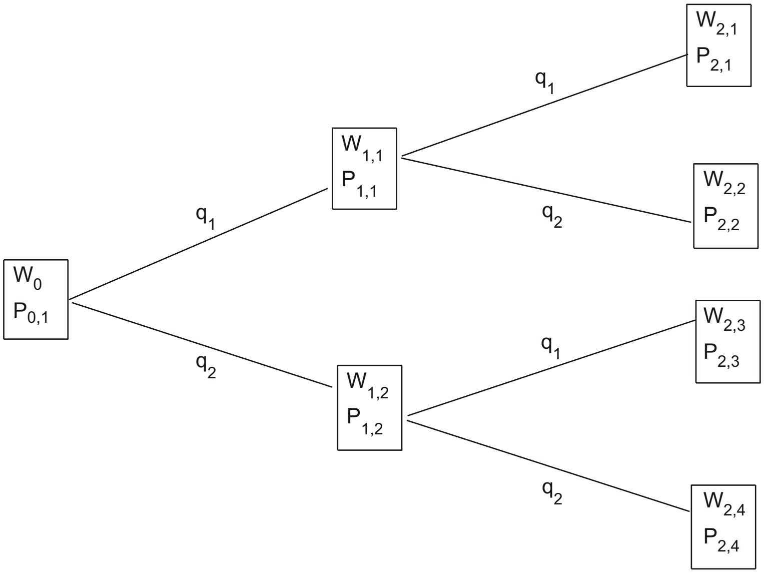


Figure 1 A binary tree with two periods.

In a mathematical formula, the probability of scenario j at time $t + 1$ is

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

and the wealth at scenario j and time $t + 1$ is

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

for $1 \le j \le m^{t+1}$ and $0 \le t < T$. Here, $W_{0,1} = W_0$ is a given initial wealth, $P_{0,1} = 1$, $\mathrm{mod}(j,\, m)$ is the remainder of division of j by m, and $[(j-1)/m]$ is the quotient of division of $(j-1)$ by m. The goal is to choose optimal bond allocations $B_{t,j}$ and stock allocations $S_{t,j}$ to maximize the expected terminal utility, i.e.,

$$\max \sum_{j=1}^{m^T} (P_{T,j} \cdot u(W_{T,j})). \tag{9}$$

We should add $B_{t,j} \ge 0$ and $S_{t,j} \ge 0$ for all t, j as bound constraints in the tree model, if neither shorting stock or borrowing bond is allowed. This tree method includes all possible scenarios with their assigned probabilities.

The disadvantage of the tree method is that when m or T is large, the problem size will exponentially increase and it will not be feasible for a solver to find an accurate solution.

In contrast, dynamic programming algorithms have no such disadvantage. As with the growth model example, the cases where we can solve the portfolio problem exactly can be used to evaluate the quality of our numerical dynamic programming methods.

Both of these examples are simple one-dimensional problems. Our examples below will also discuss solutions to multidimensional versions of both the growth model and the portfolio model.

4. TOOLS FROM NUMERICAL ANALYSIS

The previous section outlined the basic numerical challenges. In this section, we review the tools from numerical analysis that we use to produce stable and efficient algorithms. There are three main components in numerical dynamic programming: optimization, approximation, and numerical integration.

4.1 Optimization

For value function iteration, the most time-consuming portion is the optimization step. There are N_t optimization tasks at time t in Algorithm 1, one for each approximation node. If the number of value function iterations is T, then the total number of optimization tasks is $\sum_{t=1}^{T} N_t$. All these optimization tasks are relatively small problems with a small number of choice variables. Algorithm performance depends on how quickly these problems are solved.

If the value function approximation is not smooth, then the objective function of the optimization problem in the maximization step is not smooth, forcing us to use methods that can solve nonsmooth problems. If the value function approximation is smooth, we can use Newton's method and related methods for constrained nonlinear optimization problems, which have a locally quadratic convergence rate.

We often use NPSOL (Gill et al., 1994), a set of Fortran subroutines for minimizing a smooth function subject to linear and nonlinear constraints. The NPSOL libraries may be called from a driver program in Fortran, C/C++, or MATLAB. NPSOL is an appropriate optimization solver for dynamic programming applications in economics and finance, since the optimization tasks in numerical dynamic programming are small-size smooth problems.

4.2 Numerical Integration

In the objective function of the Bellman equation, we often need to compute the conditional expectation of $V(x^+)$. When the random variable is continuous, we have to use numerical integration to compute the expectation. Gaussian quadrature rules are often applied in computing the integration.

4.2.1 Gauss-Hermite Quadrature

In the expectation operator of the objective function of the Bellman equation, if the random variable has a normal distribution, then it will be good to apply the Gauss-Hermite quadrature formula to compute the numerical integration. That is, if we want to compute $\mathbb{E}\{f(Y)\}$ where Y has a distribution $\mathcal{N}(\mu, \sigma^2)$, then

$$\begin{aligned}\mathbb{E}\{f(Y)\} &= (2\pi\sigma^2)^{-1/2} \int_{-\infty}^{\infty} f(y) e^{-(y-\mu)^2/(2\sigma^2)} dy \\ &= (2\pi\sigma^2)^{-1/2} \int_{-\infty}^{\infty} f(\sqrt{2}\,\sigma\, x + \mu) e^{-x^2} \sqrt{2}\sigma\, dx \\ &\doteq \pi^{-\frac{1}{2}} \sum_{i=1}^{m} \omega_i f(\sqrt{2}\sigma x_i + \mu),\end{aligned}$$

where ω_i and x_i are the Gauss-Hermite quadrature with m weights and nodes over $(-\infty, \infty)$. If Y is log normal, i.e., $\log(Y)$ has a distribution $\mathcal{N}(\mu, \sigma^2)$, then we can assume that $Y = e^X$, where $X \sim \mathcal{N}(\mu, \sigma^2)$, thus

$$\mathbb{E}\{f(Y)\} = \mathbb{E}\{f(e^X)\} \doteq \pi^{-\frac{1}{2}} \sum_{i=1}^{m} \omega_i f\left(e^{\sqrt{2}\sigma x_i + \mu}\right).$$

See Judd (1998) for more details.

If we want to compute a multidimensional integration, we could apply the product rule. For example, suppose that we want to compute $\mathbb{E}\{f(X)\}$, where X is a random vector with multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ over $\mathbb{R}^n$, where μ is the mean column vector and Σ is the covariance matrix, then we could do the Cholesky factorization first, i.e., find a lower triangular matrix L such that $\Sigma = LL^\top$. This is feasible as Σ must be a positive semi-definite matrix from the covariance property.

4.3 Approximation

An approximation scheme consists of two parts: basis functions and approximation nodes. Approximation nodes can be chosen as uniformly spaced nodes, Chebyshev nodes, or some other specified nodes. From the viewpoint of basis functions, approximation methods can be classified as either spectral methods or finite element methods. A spectral method uses globally nonzero basis functions $\phi_j(x)$ such that $\hat{V}(x; \mathbf{b}) = \sum_{j=0}^{n} b_j \phi_j(x)$ 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, 2013). In contrast, a finite element method uses locally basis functions $\phi_j(x)$ that are nonzero over subdomains of the approximation domain. Examples of finite element methods include piecewise linear interpolation, cubic splines, and B-splines. See Cai (2010), Cai and Judd (2010), and Judd (1998) for more details.

4.3.1 Chebyshev Polynomial Approximation

Chebyshev polynomials on $[-1,1]$ are defined as $\mathcal{T}_j(z) = \cos(j\cos^{-1}(z))$. Economics problems typically live on an interval $[x_{\min}, x_{\max}]$; if we let

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

z is the original node,
x is the adapted node

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

$$\hat{V}(x; \mathbf{b}) = \sum_{j=0}^{n} b_j \mathcal{T}_j(Z(x)), \tag{10}$$

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 (10) can be easily computed by the following formula,

$$b_0 = \frac{1}{m} \sum_{i=1}^{m} v_i,$$

$$b_j = \frac{2}{m} \sum_{i=1}^{m} v_i \mathcal{T}_j(z_i), \quad j = 1, \ldots, n. \tag{11}$$

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 (10) with the coefficients given by (11) becomes Chebyshev polynomial interpolation (which is a Lagrange interpolation), as $\hat{V}(x^i; \mathbf{b}) = v_i$, for $i = 1, \ldots, m$.

4.3.2 Multidimensional Complete Chebyshev Polynomial Approximation

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

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

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

$$\hat{V}_n(x; \mathbf{b}) = \sum_{0 \le |\alpha| \le n} b_\alpha \mathcal{T}_\alpha \left(Z(x) \right), \tag{12}$$

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 $\mathbb{R}^d$.

4.3.3 Shape-Preserving Chebyshev Interpolation

One problem for Chebyshev interpolation is the absence of shape-preservation in the algorithm. To solve this, Cai and Judd (2013) create an optimization problem that modifies the Chebyshev coefficients so that concavity and monotonicity of the value function will be preserved. We begin with the Lagrange data $\{(x_i, v_i) : 1 \le i \le m\}$ generated by the maximization step of Algorithm 1, where x_i are the approximation nodes and v_i is the value of the unknown function at x_i. If theory tells us that the true value function is strictly increasing and concave, then add constraints to the fitting criterion that will impose shape restrictions.

Specifically, we approximate the value function using the functional form

$$\hat{V}(x; \mathbf{b}) = \sum_{j=0}^{n} \left(b_j^+ - b_j^- \right) \mathcal{T}_j \left(Z(x) \right), \tag{13}$$

where we replaced b_j in the Eq. (10) by $b_j^+ - b_j^-$ with $b_j^+, b_j^- \ge 0$, and we use the Chebyshev nodes x_i as approximation nodes. We choose some points $y_{i'}$ ($i' = 1, \ldots, m'$), called shape nodes, and impose the requirement that $\hat{V}(x; \mathbf{b})$ satisfies the shape conditions at the shape nodes. We want to choose the parameters $\mathbf{b}$ to minimize approximation errors but also satisfy the shape conditions. We can get a perfect fit and satisfy shape conditions if we allow n to be sufficiently large, but the problem may have too many solutions. We can be sure to get a shape-preserving Chebyshev interpolant by adding enough shape-preserving constraints and using a sufficiently high degree (bigger than $(m-1)$) polynomial, but we again could have multiple solutions and end up with a more complex polynomial than necessary.

To allow for the flexibility necessary to have both interpolation and shape properties, we penalize the use of high-order polynomials. Therefore, we solve the following linear programming problem:

$$
\begin{aligned}
\min_{b_j, b_j^+, b_j^-} \quad & \sum_{j=0}^{m-1} (b_j^+ + b_j^-) + \sum_{j=m}^{n} (j+1-m)^2 (b_j^+ + b_j^-), \\
\text{s.t.} \quad & \sum_{j=0}^{n} b_j \mathcal{T}_j'(y_{i'}) > 0, \quad i' = 1, \ldots, m', \\
& \sum_{j=0}^{n} b_j \mathcal{T}_j''(y_{i'}) < 0, \quad i' = 1, \ldots, m', \\
& \sum_{j=0}^{n} b_j \mathcal{T}_j(z_i) = v_i, \quad i = 1, \ldots, m, \\
& b_j - \hat{b}_j = b_j^+ - b_j^-, \quad j = 0, \ldots, m-1, \\
& b_j = b_j^+ - b_j^-, \quad j = m, \ldots, n, \\
& b_j^+, b_j^- \geq 0, \quad j = 1, \ldots, n,
\end{aligned}
\tag{14}
$$

where $z_i = -\cos((2i-1)\pi/(2m)) = Z(x_i)$ for $i = 1, \ldots, m$.

This problem includes interpolation among the constraints as well as the shape conditions, but chooses the polynomial with the smallest total weighted penalty, and is biased toward low-degree polynomials since a higher degree term is penalized more. The expression $b_j^+ - b_j^-$ represents b_j with $b_j^+, b_j^- \geq 0$, implying $|b_j| = b_j^+ + b_j^-$. The simple Chebyshev interpolation coefficients $\hat{b}_j$ give us a good initial guess. Therefore, we actually solve for the deviations of the Chebyshev coefficients from the simple Chebyshev interpolation coefficients.

The $y_{i'}$ are pre-specified shape nodes in $[-1, 1]$ for shape-preserving constraints. We often need to use more shape points than just the m approximation nodes since polynomial approximation need not preserve shape. There is no obvious best way to choose these points. One logical possibility is to use Chebyshev nodes corresponding to the zeroes of a degree $m' > m$ Chebyshev polynomial; however, we have no reason to think this is the best. The strong approximation properties of Chebyshev interpolation do not apply directly since shape-preservation is a one-sided inequality condition whereas Chebyshev interpolation is excellent for $\mathcal{L}^\infty$ approximation, a two-sided concept. Another choice, one that we use in our examples, is to use $m' > m$ equally spaced points. For any method we use, we may not know how many we need when we begin, so one must test the resulting solution on many more points, and increase the set of shape nodes if shape has not been preserved. As long as the value function has bounded derivatives, it is obvious that there is some finite number of shape constraints that will impose shape.

Moreover, the interpolation constraints imply that $n+1$, the number of Chebyshev polynomials used in the value function approximation, needs to be greater than the number of interpolation nodes since we need to satisfy m interpolation equality constraints and $2m'$ shape-preserving constraints in (14).

4.3.4 Shape-Preserving Hermite Interpolation

The shape-preserving Chebyshev interpolation imposes many additional shape-preserving constraints in the fitting problem and are computationally more demanding than desirable. There has been much effort developing shape-preserving and Hermite interpolation; see, for example, the survey paper in Goodman (2001). Most methods produce splines and are global, with all spline parameters depending on all the data. Judd and Solnick (1994) applied Schumaker shape-preserving polynomial splines (Schumaker, 1983) in optimal growth problems, but Schumaker splines are costly because they require creating new nodes each time a value function is constructed.

Cai and Judd (2012a) present an inexpensive shape-preserving rational function spline Hermite interpolation for a concave, monotonically increasing function. Suppose we have the Hermite data $\{(x_i, v_i, s_i) : i = 1, \ldots, m\}$, where x_i are the approximation nodes, v_i is the value of the unknown function at x_i, and s_i is its slope at x_i. With these data, we approximate the value function on the interval $[x_i, x_{i+1}]$ with

$$\hat{V}(x; \mathbf{b}) = b_{i1} + b_{i2}(x - x_i) + \frac{b_{i3}b_{i4}(x - x_i)(x - x_{i+1})}{b_{i3}(x - x_i) + b_{i4}(x - x_{i+1})}, \tag{15}$$

for $x \in [x_i, x_{i+1}]$, where

$$\begin{aligned} b_{i1} &= v_i, \\ b_{i2} &= \frac{v_{i+1} - v_i}{x_{i+1} - x_i}, \\ b_{i3} &= s_i - b_{i2}, \\ b_{i4} &= s_{i+1} - b_{i2}, \end{aligned} \tag{16}$$

for $i = 1, \ldots, m-1$. $\hat{V}(x; \mathbf{b})$ is obviously $\mathcal{C}^{\infty}$ on each interval (x_i, x_{i+1}), and $\mathcal{C}^1$ globally.

This is a local method because the rational function interpolant on each interval $[x_i, x_{i+1}]$ depends only on the level and slope information at the endpoints. Moreover, $\hat{V}(x; \mathbf{b})$ is shape-preserving. If the data is consistent with a concave increasing value function, i.e., $s_i > b_{i2} > s_{i+1} > 0$, then straightforward computations show that $\hat{V}'(x; \mathbf{b}) > 0$ and $\hat{V}''(x; \mathbf{b}) < 0$ for all $x \in (x_i, x_{i+1})$, that is, it is increasing and concave in the interval (x_i, x_{i+1}). It is also cheaply computed since the approximation on each interval depends solely on the data at its endpoints. This approach does not require adding new nodes to the spline or the determination of free parameters, features that are common in the shape-preserving polynomial spline literature.

5. SHAPE-PRESERVING DYNAMIC PROGRAMMING

Algorithm 1 is a general method for solving deterministic dynamic programming problems, but it may fail. Theory tells us that if $V_{t+1}(x)$ is concave and monotone increasing then $V_t(x)$ is also concave and monotone increasing. However, this may fail in Algorithm 1. Theory assumes that we solve the maximization step at each state but Algorithm 1 solves the maximization step at only a finite number of states and produce a finite amount of Lagrange data $\{(x_i, v_i) : i = 1, \ldots, m_t\}$. This data may be consistent with concavity, but many methods of fitting a curve to the data will produce approximations for $V_t(x)$ that violate either monotonicity or concavity, or both. If $V_t(x)$ is not concave or monotone increasing, then those errors will produce errors when we compute $V_{t-1}(x)$. These problems may create significant errors in approximating the value functions as we iterate backward in time. This is not just a theoretical possibility; an example in Section 5.1 illustrates how these problems can arise easily. In any case, if the value function approximations violate basic shape properties that we know are satisfied by the true solution, we know that we have bad approximations.

This possibly explains the tendency of economists to use piecewise linear approximations of value functions since piecewise linear approximations automatically preserve shape. While this may solve the shape problems, it causes other problems. If one uses piecewise linear approximations, then one needs to use many nodes to construct a good approximation, and the optimization problems in Algorithm 1 have nondifferentiable objective functions, a feature that rules out the use of fast Newton-type solvers. The alternatives, such as bisection, will be much slower. Also, the piecewise linear approximation approach only works for one-dimensional problems.

Dynamic programming problems in economics often make assumptions that imply monotone increasing, concave, and $\mathcal{C}^3$ value functions. It is natural to impose those properties on the value function approximations in Algorithm 1. The optimization step will be a smooth convex optimization problem for which it is easy to find the global optimum.

5.1 Application in Optimal Growth Problems

We use the following numerical examples of the finite-horizon optimal growth model (4) to illustrate the importance of the shape-preserving property. In the following examples, we let $\alpha = 0.25$, $\beta = 0.99$, $\gamma = 8$, $\eta = 1$, $A = (1-\beta)/(\alpha\beta)$, and $T = 20$. Let the range of k be $[0.1, 1.9]$, i.e., $\underline{k} = 0.1$ and $\bar{k} = 1.9$. And we choose $\epsilon = 10^{-6}$ in the model (4). The production function is $f(k, l) = Ak^{\alpha} l^{1-\alpha}$, and the utility function is a power utility with the following form

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

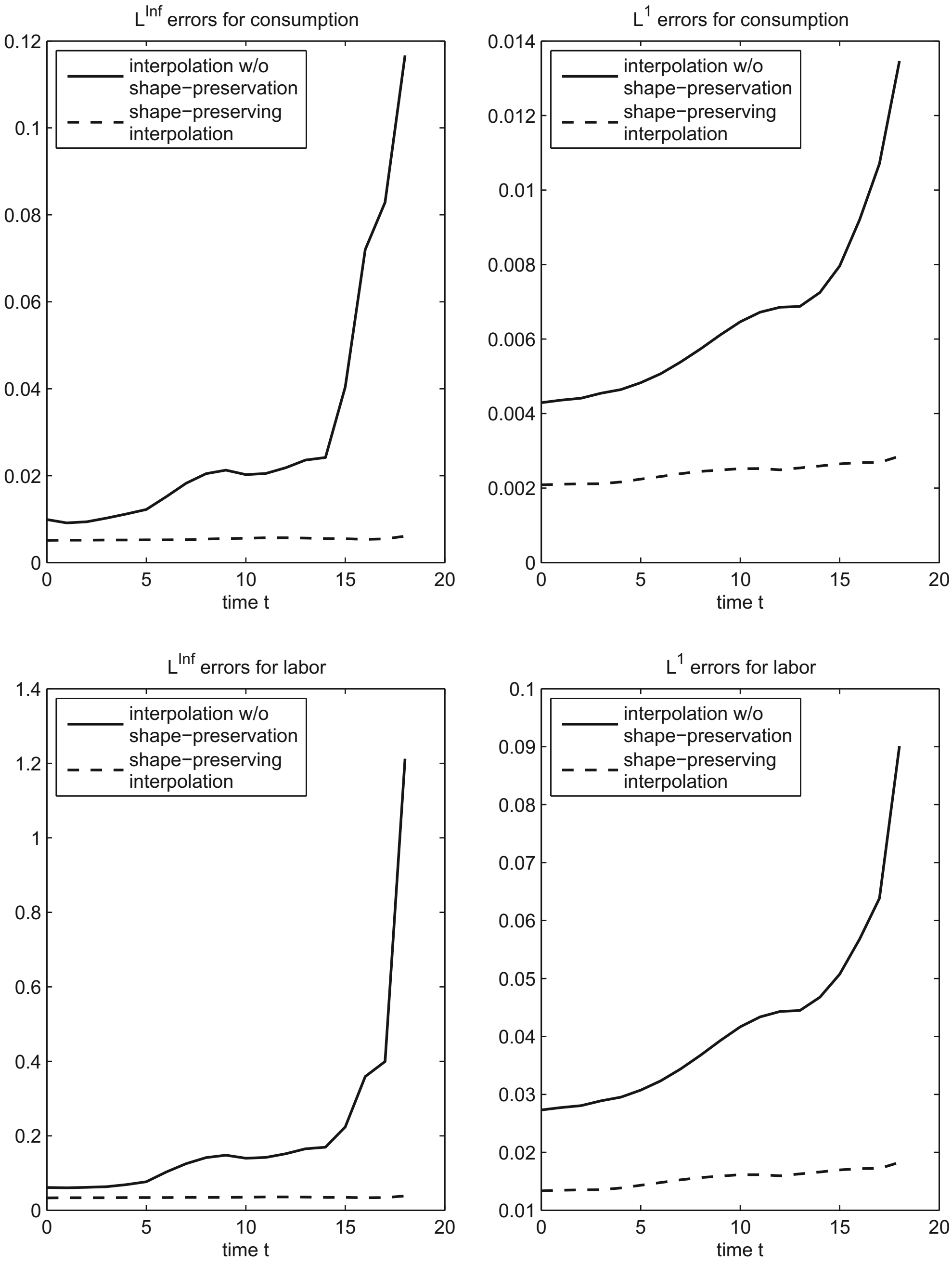


Figure 2 Errors of numerical dynamic programming with Chebyshev interpolation with/without shape-preservation for growth problems.

Thus the steady state of the infinite-horizon deterministic optimal growth problems is $k_{ss} = 1$ while the optimal consumption and the optimal labor supply at k_{ss} are, respectively, $c_{ss} = A$ and $l_{ss} = 1$. Moreover, the utility at the steady state is 0 and then the true value function at the steady state is also 0. This normalization of the typical power utility from the economic literature not only helps avoid scaling issues but also gives us a simple criterion to check if a numerical solution is close to the true solution.

We choose the terminal value function as

$$V_T(k) = \frac{u(f(k,1),1)}{1-\beta}.$$

We see that the terminal value function is smooth and concave, and the optimal controls will not be binding at least at the next-to-the-last stage $t = T - 1$. Thus, it is supposed that polynomial approximation methods could approximate the value functions well. We use the solutions given by directly applying SNOPT (Gill et al., 2005) in the model (4) as the true solutions.

Figure 2 illustrates how Chebyshev interpolation without shape-preservation produces bad approximations. Figure 2 contains four graphs corresponding to combinations of $\mathcal{L}^\infty$ and $\mathcal{L}^1$ norms with the controls, consumption, and labor supply. Each graph contains two lines; the solid line displays errors for Chebyshev interpolation without shape-preservation, and the broken line displays errors with shape-preservation. Each line shows the relative errors of consumption or labor supply using either the $\mathcal{L}^\infty$ or the $\mathcal{L}^1$ norm. Shape was imposed at $m' = 20$ equally spaced nodes in (14).

Figure 2 first shows that the errors are substantial when we ignore shape constraints. The errors are particularly large for later periods, and do decrease as we iterate backwards in time but they do not disappear. This example is a relatively easy problem, with infinitely smooth utility and production functions.

The second conclusion from Figure 2 is that shape-preservation substantially reduces the errors. Furthermore, the errors are uniformly small across time. The functional form of the approximation is a degree-9 polynomial for both methods in Figure 2; hence, the problem when we ignore shape constraints is not that there is no good degree-9 polynomial approximation of the value function. The only difference between the two procedures is the imposition of shape constraints, constraints that we know are satisfied by the true value function.

5.2 Application in Multistage Portfolio Optimization Example

We use the multistage portfolio optimization model (7) with one stock and one bond available for investment to show the shape-preservation is even more crucial when there is a kink in the optimal solutions. We assume that the number of periods is $T = 6$, the bond has a risk-free return $R_f = 1.04$, and the stock has a discrete random return

$$R = \begin{cases} 0.9, \text{ with probability } 1/2, \\ 1.4, \text{ with probability } 1/2. \end{cases} \tag{17}$$

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

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

with $\gamma = 2$ and $K = 0.2$ so that the terminal wealth should be always bigger than 0.2. Moreover, we assume that borrowing or shorting is not allowed in this example, i.e., $B_t \geq 0$ and $S_t \geq 0$ for all t.

Since the terminal utility function is $u(W_T) = (W_T - K)^{1-\gamma}/(1-\gamma)$, we know that the terminal wealth W_T must be always larger than K. It follows that we should have $W_t > KR_f^{t-T}$. Thus, since shorting or borrowing is not allowed and R is bounded, we choose the ranges $[\underline{W}_t, \overline{W}_t]$ for approximating value functions as

$$\begin{aligned} \underline{W}_{t+1} &= \max\left\{\min(R)\underline{W}_t, KR_f^{t-T} + \varepsilon\right\}, \\ \overline{W}_{t+1} &= \max(R)\overline{W}_t, \end{aligned} \tag{18}$$

with a given initial wealth bound $[\underline{W}_0, \overline{W}_0]$, where $\varepsilon > 0$ is a small number.

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

$$\begin{aligned} \left[\underline{W}_1, \ldots, \underline{W}_6\right] &= [0.81, 0.729, 0.656, 0.590, 0.531, 0.478], \\ \left[\overline{W}_1, \ldots, \overline{W}_6\right] &= [1.54, 2.156, 3.018, 4.226, 5.916, 8.282]. \end{aligned}$$

We see that the ranges are expanding exponentially along time t. If we use a fixed range along time t in our numerical dynamic programming algorithms, then it will definitely reduce the accuracy of solutions. So here we choose the above ranges at times $t = 0, 1, \ldots, 5$.

5.2.1 Numerical Results of Shape-Preserving Rational Spline Hermite Interpolation

We use the shape-preserving rational function spline Hermite interpolation (15) to solve the multistage portfolio optimization problem (7), and compare it with earlier methods. To evaluate the accuracy of our method, we compare it to the true solution. The value function has no closed-form expression because of the borrowing constraints. An example with a closed-form solution would have been too easy for our method to solve. The borrowing constraint makes this more challenging because the bond strategy has a kink at the largest wealth where it binds. However, we can compute the true solution for any

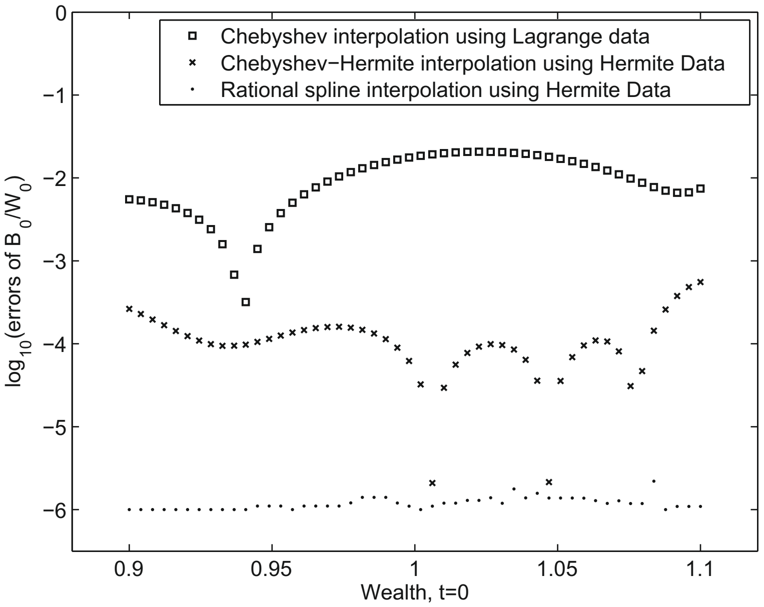


Figure 3 Errors of optimal bond allocations from numerical dynamic programming.

initial wealth using the tree method (9). The tree method solves for the state-contingent values of all variables at all nodes in the decision tree. We solve the tree model using MINOS (Murtagh and Saunders, 1982) in AMPL code (Fourer et al., 1990) via the NEOS server (Czyzyk et al., 1998). We use the true solution to measure the accuracy of our dynamic programming algorithm and compare it with the accuracy of other methods. The presence of a borrowing constraint also means we should approximate the value function, which will be $\mathcal{C}^2$, not the policy function which may only be $\mathcal{C}^0$. Polynomial approximation theory tells us to focus on approximating the smoother function.

Figure 3 shows relative errors for bond allocations of alternative dynamic programming algorithms. The squares are errors of solutions of dynamic programming with Chebyshev interpolation using Lagrange data, the x-marks are errors of dynamic programming with Chebyshev-Hermite interpolation using Hermite data, and the solid points are errors of dynamic programming with the rational function spline interpolation using Hermite data. All the computational results are given by MINOS (Murtagh and Saunders, 1982) in AMPL (Fourer et al., 1990) via the NEOS server (Czyzyk et al., 1998). For dynamic programming with Chebyshev interpolation or dynamic programming with Chebyshev-Hermite interpolation, we use $m = 10$ Chebyshev nodes and degree-9 or degree-19 Chebyshev polynomials, respectively. For the rational function spline interpolation, we use $m = 10$ equally spaced nodes.

Table 1 Errors of optimal bond allocations for various γ.

γ	Number of approximation nodes	Errors at time $t = 1$
2	10	1.1×10^{-6}
4	20	7.3×10^{-4}
	40	1.1×10^{-4}
8	20	3.9×10^{-3}
	40	5.3×10^{-4}

We see that the errors are about $O(10^{-1})$ or $O(10^{-2})$ for Chebyshev interpolation using Lagrange data, while they are about $O(10^{-3})$ or $O(10^{-4})$ for Chebyshev-Hermite interpolation (Cai and Judd, 2012b) using Hermite data. However, the errors of the rational function spline Hermite interpolation is always about $O(10^{-6})$, showing that it has the best performance for approximating value functions.

Table 1 lists numerical errors of optimal bond allocations from dynamic programming with the rational function spline interpolation, for various values of γ. We see that even for large γ, the solutions from dynamic programming with the rational function spline interpolation are still good.

Our new approximation method was always as fast as any of the other algorithms. Therefore, the shape-preserving rational function spline Hermite interpolation is reliable and often substantially better than other approximation methods.

5.2.2 Other Shape-preserving Methods

There are many methods for preserving shape (see Goodman, 2001) but many are not suitable for our purposes. The one-dimensional Schumaker shape-preserving interpolation method (Schumaker, 1983) was applied to dynamic programming in Judd and Solnick (1994) and Cai (2010). However, the approximation is more complex than the ones discussed above, and is at best $\mathcal{C}^1$ whereas Newton solvers really prefer $\mathcal{C}^2$ or smoother value function approximations. Wang and Judd (2000) applied a bivariate shape-preserving spline interpolation method (Costantini and Fontanella, 1990) in numerical dynamic programming to solve a savings allocation problem. However, the bivariate method only preserved shape along the coordinate axes, whereas the shape-preserving Chebyshev interpolation method (Cai and Judd, 2013) can be generalized to higher dimensions and impose shape restrictions in any direction. The mathematics literature on shape-preserving approximation is mostly focused on one- or two-dimensional problems, forcing economists to develop their own methods when solving higher dimensional dynamic programming problems.

6. PARALLELIZATION

Many dynamic programming problems in economics involve many states, and solving them will face the "curse of dimensionality." Even if one uses approximation and quadrature methods that avoid the curse of dimensionality, dynamic programming problems with many states are expensive to solve. If parallelization can be used, it is the natural way to make otherwise intractable problems tractable. Many modern computer systems now offer researchers parallel computing tools. Fortunately, dynamic programming problems do have a structure that facilitates the use of parallelization.

Cai et al. (2013b) implement a parallel dynamic programming algorithm on a computational grid consisting of loosely coupled processors, possibly including clusters and individual workstations. The grid changes dynamically during the computation, as processors enter and leave the pool of workstations. The algorithm is implemented using the Master-Worker library running on the HTCondor grid computing platform. We implement value function iteration for large optimal growth problems. We present examples that solve in hours on HTCondor but would take weeks if executed on a single workstation. The use of HTCondor can increase a researcher's computational productivity by at least two orders of magnitude.

In the value function iteration, a set of discrete and approximation nodes will be chosen and the period t value function at those nodes will be computed and then we can use some approximation methods to approximate the value function. For every approximation node, there is a time-consuming optimization problem to be solved. Moreover, these optimization problems are independent, allowing them to be solved efficiently in parallel.

6.1 The Variety of Parallel Programming Architectures

There are three basic approaches to massive parallelism. Supercomputing is a well-known example of massive parallelism. Supercomputers combine large numbers of identical processors with specialized communication hardware that allows for rapid communication among processors. This is called high-performance computing (HPC). Supercomputers are able to solve some very large problems at high efficiency. However, attaining these speeds puts rigid requirements on problems. Users of supercomputers are generally given a fixed block of processors for a fixed amount of time. This structure requires that users reserve supercomputer time, and the lag time between requests and the actual allocation will increase with the number of desired processors and requested time. Moreover, economists face substantial bureaucratic hurdles in getting access to supercomputer time because the people who control supercomputers impose requirements that are met by few economists. In particular, the authors have been told that US Department of Energy supercomputers available to the general scientific community are not available to economists who want to analyze policy issues, such as taxation problems.

Second, there is high-throughput computing (HTC) which may be slower but is a paradigm with much greater flexibility and lower cost. HTCondor is an example of HTC and a valuable alternative to HPC. The HTCondor system is an open-source software framework for distributed parallelization of computationally intensive tasks on a cluster of computers. HTCondor accumulates a set of desired tasks from users, and then allocates them to those computers that are not being used at the moment. HTCondor acts as a management tool for identifying, allocating, and managing available resources to solve large distributed computations. For example, if a workstation on a network is currently unused, HTCondor will detect that fact, and send it a task. HTCondor will continue to use that workstation until a higher-priority user (such as a student sitting at the keyboard) appears, at which time HTCondor ends its use of the workstation. This is called "cycle scavenging" and allows a system to take advantage of essentially free computing time. The marginal social cost of CPU time used in HTCondor is essentially zero because it is using CPU time that otherwise would go unused. HTCondor manages the number of processors being used in response to processor availability and the needs of the computational procedure. HTC is opportunistic, utilizing any resource that becomes available and does not force the user to make reservations. The disadvantage of HTC is that interprocessor communication will be only as fast as communication among computers in a cluster, a speed considerably slower than that in supercomputers. While this does limit the amount of parallelization that can be exploited, HTC environments can still efficiently use hundreds of processors for many problems.

The HTCondor team at the University of Wisconsin-Madison has developed several "flavors" of HTCondor, each fine-tuned for some specific type of parallel programming. For our dynamic programming problems, we used the HTCondor Master-Worker (MW) system. The HTCondor MW system consists of two entities: a master process and a cluster of worker processes. The master process decomposes the problem into small tasks and puts those tasks in a queue. Each worker process first examines the queue, takes the "top" problem off the queue, and solves it. The worker then sends the results to the master, examines the queue of unfinished tasks, and repeats this process until the queue is empty. The workers' execution is a simple cycle: take a task off master's queue, do the task, and then send the results to the master. While the workers are solving the tasks, the master collects the results and puts new tasks on the queue. This is a file-based, remote I/O scheme that serves as the message-passing mechanism between the master and the workers.

Third, there is grid computing which spreads work across computers connected only by the Internet. While the authors are not aware of any economics applications of grid computing, it is used extensively in the sciences. See BOINC (http://boinc.berkeley.edu) for a discussion of grid computing applied to scientific projects.

Based on our experiences, we believe that all three forms of massive parallelism can be used to solve large dynamic programming problems. Our discussion below will focus on our use of HTCondor, but the same basic approach will work on both supercomputers and grid computing.

6.2 Parallel Dynamic Programming

The numerical dynamic programming algorithms can be applied easily in the HTCondor MW system for dynamic programming problems with multidimensional continuous and discrete states. To solve these problems, numerical dynamic programming algorithms with value function iteration have the maximization step that is mostly time-consuming in numerical dynamic programming. Equation (3) in Algorithm 2 computes $v_{i,j}$ for each approximation point x^i in the finite set $\mathbb{X}_t \subset \mathbb{R}^n$ and each discrete state vector $\theta^j \in \Theta$, where N_t is the number of points of $\mathbb{X}_t$ and D is the number of points of Θ, resulting in $N_t \times D$ small maximization problems. If the $N_t \times D$ is large, as it is for high-dimensional problems, then these maximization steps will consume most of the time used in any algorithm. However, these $N_t \times D$ small-size maximization problems can be naturally parallelized in the HTCondor MW system, in which one or several maximization problem(s) could be treated as one task.

We first present an example where we parallelize the problem across the discrete states. After that presentation, we will indicate how to parallelize in the continuous dimensions of the state space.

When D is large but the number of approximation nodes, N_t, is of medium size, it is natural to separate the $N_t \times D$ maximization problems into D tasks, where each task corresponds to a discrete state vector θ^j and all continuous state nodes set $\mathbb{X}_t$. Algorithm 3 is the architecture for the master processor, and Algorithm 4 is the corresponding architecture for the workers.

Algorithm 3. Parallel Dynamic Programming with Value Function Iteration for the Master

```
Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^{d'}$. Set $\mathbf{b}^T$ as the
    parameters of the terminal value function. For $t = T-1, T-2, \ldots, 0$, iterate
    through steps 1 and 2.
Step 1. Separate the maximization step into $D$ tasks, one task per $\theta \in \Theta$. Each task
    contains parameters $\mathbf{b}^{t+1}$, stage number $t$, and the corresponding task identity for
    some $\theta^j$. Then send these tasks to the workers.
Step 2. Wait until all tasks are done by the workers. Then collect parameters $\mathbf{b}_j^t$ from
    the workers, for all $1 \leq j \leq D$, and let $\mathbf{b}^t = \{\mathbf{b}_j^t : 1 \leq j \leq D\}$.
```

Algorithm 4. Parallel Dynamic Programming with Value Function Iteration for the Workers

Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^{d'}$ and the probability transition matrix $P = \left(p_{j,j'}\right)_{D\times D}$ where $p_{j,j'}$ is the transition probability from $\theta^j \in \Theta$ to $\theta^{j'} \in \Theta$ for $1 \leq j, j' \leq D$. Choose a functional form for $\hat{V}(x, \theta; \mathbf{b})$ for all $\theta \in \Theta$.

Step 1. Get parameters $\mathbf{b}^{t+1}$, stage number t, and the corresponding task identity for one $\theta^j \in \Theta$ from the master, and then choose the approximation grid, $\mathbb{X}_t = \{x_t^i : 1 \leq i \leq N_t\} \subset \mathbb{R}^d$.

Step 2. For this given θ^j, compute

$$v_{i,j} = \max_{a\in\mathcal{D}(x^i,\theta^j,t)} u(x^i, \theta^j, a) + \beta\mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\},$$

for each $x^i \in \mathbb{X}_t$, $1 \leq i \leq N_t$, where the next-stage discrete state $\theta^+ \in \Theta$ is random with probability mass function $\mathbb{P}(\theta^+ = \theta^{j'} \mid \theta^j) = p_{j,j'}$ for each $\theta^{j'} \in \Theta$, and x^+ is the next-stage state transition from x^i and may be also random.

Step 3. Using an appropriate approximation method, compute $\mathbf{b}_j^t$ such that $\hat{V}(x, \theta^j; \mathbf{b}_j^t)$ approximates $\{(x^i, v_{i,j})$: $1 \leq i \leq N_t\}$, i.e., $v_{i,j} \approx \hat{V}(x^i, \theta^j; \mathbf{b}_j^t)$ for all $x^i \in \mathbb{X}_t$.

Step 4. Send $\mathbf{b}_j^t$ and the corresponding task identity for θ^j to the master.

Algorithm 3 describes the master's function. Suppose that the value function for time $t + 1$ is known, and the master wants to solve for the value function at period t. For each point $\theta \in \Theta$, the master gathers all the Bellman optimization problems associated with that θ, together with the solution for the next period's value function, and sends that package of problems to a worker processor. It does this until all workers are working on some such package. When the master receives the solutions from a worker, it records those results and sends that worker another package of problems not yet solved. This continues until all θ specific packages have been solved, at which point the master repeats this for period $t - 1$.

Algorithm 4 describes the typical worker task. It takes the θ^j package from the master, solves the Bellman optimization problem for each node in $\mathbb{X}_t$, and computes the new value for $\mathbf{b}_j^t$, the coefficients for the value function in the θ^j dimension, and sends those coefficients back to the master.

The case where we parallelize only across the discrete dimensions is easy to implement, and is adequate if the number of available workers is small relative to the number of points in Θ. If we have access to more workers, then we will also parallelize across points in $\mathbb{X}_t$. The key difference in that case is that each worker can only compute some of the $v_{i,j}$ values needed to determine $\mathbf{b}_j^t$. One way to proceed is to send all the $v_{i,j}$ values to the master which then executes the fitting step, or, if that is too demanding, the master will send that task to a worker to compute $\mathbf{b}_j^t$. See Cai et al. (2013b) for more details on this case.

Our parallelization examples of economic problems, as described above, have used only the most basic techniques for coordinating computation among processors. There are many other places where parallelization might be useful. For example, if the Bellman optimization problem corresponding to a single point (x^i, θ^j) in the state space were itself a large problem, and we had a large number of processors, then it might be useful to use a parallel algorithm to solve each such state-specific problem. There are many possible ways to decompose the big problem into smaller ones and exploit the available processors. We have discussed only the first two layers of parallelization that can be used in dynamic programming. How fine we go depends on the number of processors at our disposal and the communication times across computational units.

6.3 Application to Stochastic Optimal Growth Models

We consider a multidimensional stochastic optimal growth problem. We assume that there are d sectors, and let $k_t = (k_{t,1}, \dots, k_{t,d})$ denote the capital stocks of these sectors which is a d-dimensional continuous state vector at time t. Let $\theta_t = \left(\theta_{t,1}, \dots, \theta_{t,d}\right) \in \Theta = \{\theta_t^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$ denote current productivity levels of the sectors which is a d-dimensional discrete state vector at time t, and assume that θ_t follows a Markov process with a stable probability transition matrix, denoted as $\theta_{t+1} = g(\theta_t, \xi_t)$ where ξ_t are i.i.d. disturbances. Let $l_t = \left(l_{t,1}, \dots, l_{t,d}\right)$ denote elastic labor supply levels of the sectors which is a d-dimensional continuous control vector variable at time t. Assume that the net production function of sector i at time t is $f(k_{t,i}, l_{t,i}, \theta_{t,i})$, for $i = 1, \dots, d$. Let $c_t = \left(c_{t,1}, \dots, c_{t,d}\right)$ and $I_t = \left(I_{t,1}, \dots, I_{t,d}\right)$ denote, respectively, consumption and investment of the sectors at time t. We want to find an optimal consumption and labor supply decisions such that expected total utility over a finite-horizon time is maximized, i.e.,

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

where k_0 and θ_0 are given, δ is the depreciation rate of capital, $\Gamma_{t,j}$ is the investment adjustment cost of sector j, and ζ governs the intensity of the friction, $\epsilon_t = \left(\epsilon_{t,1}, \ldots, \epsilon_{t,d}\right)$ are serially uncorrellated i.i.d. disturbances with $\mathbb{E}\{\epsilon_{t,i}\} = 0$, and $V_T(k, \theta)$ is a given terminal value function. This model is the finite-horizon version of the problems introduced in Den Haan et al. (2011), and Juillard and Villemot (2011).

6.3.1 Dynamic Programming Model

The dynamic programming formulation of the multidimensional stochastic optimal growth problem is

$$\begin{aligned} V_t(k,\theta) = \max_{c,l,I} \; & u(c,l) + \beta \mathbb{E}\left\{ V_{t+1}(k^+,\theta^+) \mid \theta \right\}, \\ \text{s.t.} \quad & k_j^+ = (1-\delta)k_j + I_j + \epsilon_j, \quad j = 1, \ldots, d, \\ & \Gamma_j = \frac{\zeta}{2} k_j \left(\frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, \ldots, d, \\ & \sum_{j=1}^{d} \left(c_j + I_j - \delta k_j \right) = \sum_{j=1}^{d} \left(f(k_j, l_j, \theta_j) - \Gamma_j \right), \\ & \theta^+ = g(\theta, \xi_t), \end{aligned}$$

for $t = 0, \ldots, T-1$, where $k = (k_1, \ldots, k_d)$ is the continuous state vector and $\theta = (\theta_1, \ldots, \theta_d) \in \Theta = \{(\vartheta_{j,1}, \ldots, \vartheta_{j,d}) \colon 1 \le j \le D\}$ is the discrete state vector, $c = (c_1, \ldots, c_d)$, $l = (l_1, \ldots, l_d)$, and $I = (I_1, \ldots, I_d)$ are control variables, $\epsilon = (\epsilon_1, \ldots, \epsilon_d)$ are i.i.d. disturbance with mean 0, and $k^+ = (k_1^+, \ldots, k_d^+)$ and $\theta^+ = \left(\theta_1^+, \ldots, \theta_d^+\right) \in \Theta$ are the next-stage state vectors. Numerically, $V(k, \theta)$ is approximated with given values at finite nodes, so the approximation is only good at a finite range. That is, the state variable must be in a finite range $[\underline{k}, \bar{k}]$, then we should have the restriction $k^+ \in [\underline{k}, \bar{k}]$. Here $\underline{k} = (\underline{k}_1, \ldots, \underline{k}_d)$, $\bar{k} = (\bar{k}_1, \ldots, \bar{k}_d)$, and $k^+ \in [\underline{k}, \bar{k}]$ denotes that $k_i^+ \in [\underline{k}_i, \bar{k}_i]$ for all $1 \le i \le d$. Moreover, we should add $c > 0$ and $l > 0$ in the constraints.

6.3.2 Numerical Example

In the following numerical example, we see the application of parallelization of numerical dynamic programming algorithms for the dynamic programming model of the multi-dimensional stochastic optimal growth problem. We let $T = 5, \beta = 0.8, \delta = 0.025, \zeta = 0.5, [\underline{k}, \bar{k}] = [0.2, 3.0]^d, f(k_i, l_i, \theta_i) = \theta_i A k_i^{\psi} l_i^{1-\psi}$ with $\psi = 0.36$ and $A = (1-\beta)/(\psi\beta) = 1$, for $i = 1, \ldots, d$, and

$$u(c,l) = \sum_{i=1}^{d} \left[\frac{(c_i/A)^{1-\gamma} - 1}{1-\gamma} - (1-\psi) \frac{l_i^{1+\eta} - 1}{1+\eta} \right],$$

with $\gamma = 2$ and $\eta = 1$.

In this example, we let $d = 4$. So this is a dynamic programming example with four-dimensional continuous states and four-dimensional discrete states. Here we assume that the possible values of θ_i are in $\{0.85, 0.9, 0.95, 1.0, 1.05, 1.1, 1.15\}$. We assume that if the current state is i then there is a 25% chance of moving to $i - 1$, 25% chance of moving to $i + 1$, and 50% chance of staying, except at the boundaries where there is a reflecting boundary. We assume that $\theta_1^+, \ldots, \theta_d^+$ are independent of each other. In addition, we assume that $\epsilon_1, \ldots, \epsilon_4$ are i.i.d., and each ϵ_i has possible discrete values in $\{-0.01, 0.0, 0.01\}$, while their probabilities are 0.25, 0.5, and 0.25, respectively.

The continuous value function approximation is the complete degree-6 Chebyshev polynomial approximation method (12) with $7^4 = 2401$ Chebyshev nodes for continuous state variables, the optimizer is NPSOL (Gill et al., 1994), and the terminal value function is chosen as

$$V_T(k, \theta) = u(f(k, \mathbf{e}, \mathbf{e}), \mathbf{e})/(1 - \beta),$$

where $\mathbf{e}$ is the vector with 1's everywhere. Here $\mathbf{e}$ is chosen because it is the steady-state labor supply for the corresponding infinite-horizon problem and is also the average value of θ.

6.3.3 Parallelization Results

We use the master Algorithm 3 and the worker Algorithm 4 to solve the optimal growth problem. Since the number of possible values of θ_i is 7 for $i = 1, \ldots, 4$, the total number of HTCondor-MW tasks for one value function iteration is $7^4 = 2401$, and each task computes 2401 small-size maximization problems as there are 2401 Chebyshev nodes.

Under HTCondor, we assign 50 workers to do this parallel work. Table 2 lists some statistics of our parallel dynamic programming algorithm under HTCondor-MW system for the growth problem after running three value function iterations (VFI). The last line of Table 2 shows that the parallel efficiency of our parallel numerical dynamic programming method is very high (up to 98.6%) for this example. We see that the total CPU time used by all workers to solve the optimal growth problem is nearly 17 days, i.e., it will take nearly 17 wall clock days to solve the problem without using parallelism. However, it takes only 8.28 wall clock hours to solve the problem if we use the parallel algorithm and 50 worker processors.

Table 3 gives the parallel efficiency with various numbers of worker processors for this optimal growth model. We see that it has an almost linear speed-up when we add the number of worker processors from 50 to 200. We see that the wall clock time to solve the problem is only 2.26 h now if the number of worker processors increases to 200.

Parallel efficiency drops from 99% to 92% when we move from 100 processors to 200. This is not the critical fact for a user. The most important fact is that requesting 200 processors reduced the waiting time from submission to final output by 1.6 h. Focusing on the user's waiting time is one of the values of the HTC approach to parallelization.

7. DYNAMIC PORTFOLIO OPTIMIZATION WITH TRANSACTION COSTS

Any investment strategy involves dynamic management of assets, spelling out when one trades assets for other assets—rebalancing a portfolio—or for cash to finance consumption. Conventional theory assumes there are no costs to asset trades. This is not true of real markets. Even if conventional brokerage fees are small, the presence of any bid-ask spread is essentially a transaction cost since the sale price is less than the purchase price. The presence of even small transaction costs can have significant impact on investment strategies; for example, Judd et al. (2012) show that even infinitesimal transaction costs reduce bond portfolio rebalancing to nearly zero. Therefore, any examination of real-world dynamic portfolio management needs to consider these frictions.

Multistage portfolio optimization problems with transaction costs have been studied in many papers (see Abrams and Karmarkar, 1980; Boyle and Lin, 1997; Brown and Smith, 2011; Constantinides, 1976, 1979, 1986; Gennotte and Jung, 1994; Kamin, 1975; Zabel, 1973, etc.). The key insight is that transaction costs create a "no-trade region" (NTR); that is, no trading is done if the current portfolio is inside the no-trade region, and otherwise the investor trades to some point on the boundary of the no-trade region.

Multistage portfolio optimization problems with transaction costs assume that there are k risky assets ("stocks") and/or a riskless asset ("bank account" paying a fixed interest rate r) traded during the period $[0, T]$. In our discrete-time analysis, portfolio adjustments are made at time $t = 0, 1, \ldots, T-1$. Trades are made to maximize the investor's expected utility over terminal wealth (T is the terminal time) and/or consumption during $[0, T]$. If the major transaction cost is the bid-ask spread, then a proportional transaction costs is the correct case to study.

Cai (2010), Cai and Judd (2010), and Cai et al. (2013c) introduce application of numerical dynamic programming algorithms in multistage portfolio optimization problems with

Table 2 Statistics of parallel dynamic programming under HTCondor-MW for the growth problem.

Wall clock time for three VFIs	8.28 h
Total time workers were assigned	16.9 days
Average wall clock time per task	199 s
Number of (different) workers	50
Overall parallel performance	98.6%

Table 3 Parallel efficiency for various number of worker processors.

# Worker processors	Parallel efficiency (%)	Average task wall clock time (s)	Total wall clock time (h)
50	98.6	199	8.28
100	97	185	3.89
200	91.8	186	2.26

transaction costs, and showed that the method performs very well for the problems with three or more risky assets and $T \geq 6$ with general utility functions. We assume that an investor begins with some wealth W_0 and initial investment allocation x_0 accros several risky assets, and manages it so as to maximize the expected utility of wealth at time T, while there exist transaction costs at each rebalancement time. We assume a power utility function for terminal wealth, $V_T(W) = W^{1-\gamma}/(1-\gamma)$ where $\gamma > 0$ and $\gamma \neq 1$. A multistage portfolio optimization problem with transaction costs is to find an optimal portfolio x_t at each time t such that we have a maximal expected terminal utility, i.e.,

$$V_0(W_0, x_0) = \max_{x_t, 0 \leq t < T} \mathbb{E}\{V_T(W_T)\},$$

where W_T is the terminal wealth with the given initial wealth W_0 and initial allocation x_0 in the risky assets.

Let $R = (R_1, \ldots, R_n)$ be the random one-period return of n risky assets, and R_f be the return of the riskless asset. The portfolio share for asset i at the beginning of a period is denoted x_i. The state variables are the wealth W and allocations $x = (x_1, \ldots, x_n)^\top$ invested in the risky assets at the beginning of a period. The difference between wealth and the wealth invested in stocks is invested in bonds. We assume a proportional transaction cost τ for all sales and purchases of the risky assets. Let $\delta_i^+ W$ denote the amount of asset i purchased, expressed as a fraction of wealth, and let $\delta_i^- W$ denote the amount sold, where $\delta^+, \delta^- \geq 0$. Let e denote the column vector where $e_i = 1$ for all i.

The dynamic programming problem becomes

$$V_t(W, x) = \max_{\delta^+, \delta^- \geq 0} \mathbb{E}\left\{V_{t+1}(W^+, x^+)\right\},$$

where $X_i^+ \equiv R_i(x_i + \delta_i^+ - \delta_i^-)W$ is time $t+1$ wealth in asset i, $m \equiv e^\top(\delta^+ - \delta^- + \tau(\delta^+ + \delta^-))$ where mW is the change in bond holding, $W^+ \equiv e^\top X^+ + R_f(1 - e^\top x - m)W$ is time $t+1$ wealth, and $x^+ \equiv X^+/W^+$ is the vector of risky asset allocations. Given the isoelasticity of V_T, we know that $V_t(W, x) = W^{1-\gamma} g_t(x)$, for some functions $g_t(x)$, $t = 0, 1, \ldots, T-1$. The numerical task reduces to approximating the $g_t(x)$ functions.

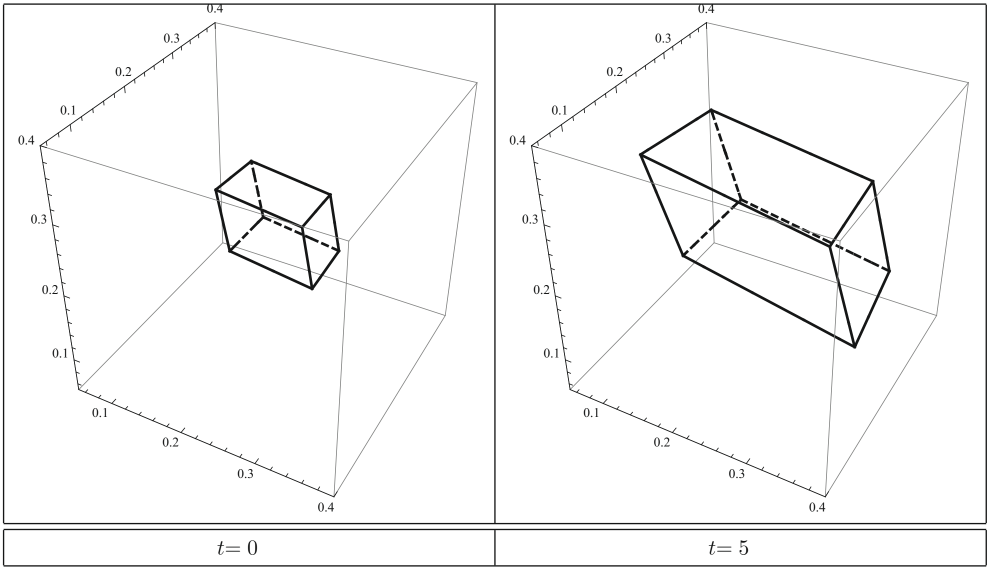
7.1 Numerical Example

We assume three stocks and one bond, where the stock returns are log-normally distributed, $\log(R) \sim \mathcal{N}(\mu, \Sigma)$, $\mu = (0.0572, 0.0638, 0.07)$, and

$$\Sigma = \begin{bmatrix} 0.0256 & 0.00576 & 0.00288 \\ 0.00576 & 0.0324 & 0.0090432 \\ 0.00288 & 0.0090432 & 0.04 \end{bmatrix}.$$

We assume a power terminal value function with $\gamma = 3.5$, a transaction cost of $\tau = 0.01$ and $R_f = 1.0408$. We use the degree-7 complete Chebyshev polynomial approximation method and a multidimensional product Gauss-Hermite quadrature rule with 9 nodes in each dimension to compute expectations. We assume a $T = 6$ investment horizon.

The key property of the solution is a no-trade region Ω_t for $t = 0, \ldots, 5$. When $x_t \in \Omega_t$, the investor will not trade at all, and when $x_t \notin \Omega_t$, the investor will trade to some point on the boundary of Ω_t. Since the value function has the form $W^{1-\gamma} g_t(x)$, the optimal portfolio rules and the "no-trade" regions Ω_t are independent of W. Figure 4 shows the no-trade regions for periods $t = 0, 5$. We see that the no-trade region grows as t approaches T. This corresponds to the intuition that an investor is not likely to adjust his portfolio if he has to pay transaction costs and the holding time is short.



t= 0

t= 5

Figure 4 No-trade regions.

8. DYNAMIC STOCHASTIC INTEGRATION OF CLIMATE AND ECONOMY

The examples presented above are simple ones that do not require the most advanced computational tools. In this section we described a far more ambitious use of numerical dynamic programming, and is a much better demonstration of the potential power of efficient dynamic programming.

There is growing concern about the impact of human activity on the climate. The global climate system is complex and its response to future increases in anthropogenic GHGs is poorly understood. Any rational policy choice must take into account the uncertainty about the magnitude and timing of climate change on economic productivity. This section describes the use of numerical dynamic programming algorithms to solve DSICE (Cai et al., 2012b, 2013a), a model of dynamic stochastic integration of climate and economy. DSICE is a DSGE extension of the DICE2007 model of William Nordhaus (Nordhaus, 2008). It is a well-known model used to examine various issues. One focus of research is estimating the expected social marginal cost of an extra ton of carbon in the atmosphere. This was, for example, the application of DICE used by IWG (2010).

The major contribution of DSICE is the ability to flexibly consider the importance of social attitudes toward risk. We know from the equity premium literature (Mehra and Prescott, 1985) that the standard, time-separable specifications of consumer preferences are incapable of modeling how people feel about risk. Kreps and Porteus (1978) have argued that there could be value in early resolution of uncertainty, and Epstein and Zin (1989) preferences have explored the implications of this for asset pricing.

Cai et al. (2012b, 2013a) build on Cai et al. (2012a) and combine standard features of DSGE models—productivity shocks, dynamic optimizing agents, and short time periods—with DICE2007, and adds uncertainty about the climate's response to greenhouse gases. Specifically, it examined the impact of "tipping points" (see also Lontzek et al., 2012). A tipping point is an event that is triggered by warming and has a permanent impact on economic productivity. Lenton et al. (2008) discuss examples of possible tipping points.

8.1 A Stochastic IAM with Epstein-Zin Preferences

This section briefly describes the DSICE model and summarizes its conclusions. See Cai et al. (2012b, 2013a), for more details.

Let $\mathbf{M}_t = (M_t^{\mathrm{AT}}, M_t^{\mathrm{UP}}, M_t^{\mathrm{LO}})^\top$ be a three-dimensional vector describing the carbon concentrations in the atmosphere, and upper and lower levels of the ocean. These concentrations evolve over time according to:

$$\mathbf{M}_{t+1} = \mathbf{\Phi}^{\mathrm{M}}\mathbf{M}_t + \left(\mathcal{E}_t\left(k_t, \mu_t\right), 0, 0\right)^\top,$$

where $\mathbf{\Phi}^{\mathrm{M}}$ is a linear transition matrix, and $\mathcal{E}_t\,(k_t, \mu_t)$ is the annual total carbon emissions dependent on the capital k_t and the fraction of potential emissions that are mitigated, μ_t. The anthropogenic sources of carbon are represented by the $\mathcal{E}_t$, which will be specified below. The DICE climate system also includes temperatures of the atmosphere and ocean, represented by $\mathbf{T}_t = (T_t^{\mathrm{AT}}, T_t^{\mathrm{LO}})^\top$. The temperatures follow the law of motion

$$\mathbf{T}_{t+1} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{T}_t + \left(\xi_1 \mathcal{F}_t\left(M_t^{\mathrm{AT}}\right), 0\right)^\top,$$

where $\mathbf{\Phi}^{\mathrm{T}}$ is a linear transition matrix. Atmospheric temperature is affected by the total radiative forcing, $\mathcal{F}_t\left(M^{\mathrm{AT}}\right)$, dependent on the atmospheric carbon concentration.

The impact of global warming on the economy is reflected by a simple damage function in DICE. Our damage function includes a term J_t which represents damage to output due to the tipping point. $J_t = 0$ if tipping has not occurred by time t, and $0 < J_t < 1$ at all times t after a tipping event. The likelihood of tipping depends on current temperature. Thus, the stochastic damage factor in DSICE is

$$\Omega\left(T_t^{\mathrm{AT}}, J_t\right) = \frac{1 - J_t}{1 + \pi_1 T_t^{\mathrm{AT}} + \pi_2 (T_t^{\mathrm{AT}})^2},$$

where the denominator represents the standard damage function from the DICE2007 model.

Capital k_t follows the rule

$$k_{t+1} = (1 - \delta)k_t + \mathcal{Y}_t(k_t, T_t^{\mathrm{AT}}, \mu_t, J_t) - c_t,$$

where c_t denotes the consumption level and $\mathcal{Y}_t$ denotes the production function, which is affected by the damage factor, $\Omega\left(T_t^{\mathrm{AT}}, J_t\right)$, the capital k_t, and the emission control rate μ_t.

8.2 Dynamic Programming with Epstein-Zin Preferences

The standard separable utility function in the finite-horizon DICE2007 class of models is

$$u(c_t, l_t) = \frac{(c_t/l_t)^{1-\psi}}{1 - \psi} l_t, \tag{19}$$

where l_t is the total labor supply (and population). DSICE assumes a social planner maximizes the present-discounted utility stream up to a terminal time T (600 years), using Epstein-Zin preferences.

The dynamic optimization problem has six continuous state variables: the capital stock, k, the three-dimensional carbon system, $\mathbf{M}$, and the two-dimensional temperature vector, $\mathbf{T}$. Furthermore, J is the discrete shock to the climate. The recursive formulation of the social planner's objective is

$$U_t\left(k, \mathbf{M}, \mathbf{T}, J\right) = \max_{c,\mu} \left\{ (1-\beta)\, u(c_t, l_t) + \beta \left[\mathbb{E}\left\{ \left(U_{t+1}\left(k^{+}, \mathbf{M}^{+}, \mathbf{T}^{+}, J^{+}\right)\right)^{1-\gamma} \right\} \right]^{\frac{1-\psi}{1-\gamma}} \right\}^{\frac{1}{1-\psi}},$$

where ψ is the inverse of the intertemporal elasticity of substitution, γ is the risk aversion parameter, and β is the discount factor. The actual risk premium will depend on interactions between ψ and γ. The special case of $\psi = \gamma$ is the time-separable specification where both parameters represent both risk aversion and the elasticity of substitution. In general, increasing γ will correspond to a greater willingness to sacrifice consumption to avoid risk. We use Epstein-Zin preferences because they are better at explaining the willingness to pay to reduce risk. This is the conclusion from the literature on the equity premium puzzle.

The dynamic optimization problem can be formulated in terms of the value function

$$V_t\left(k, \mathbf{M}, \mathbf{T}, J\right) = \frac{\left[U_t\left(k, \mathbf{M}, \mathbf{T}, J\right)\right]^{1-\psi}}{(1-\psi)\,(1-\beta)},$$

which implies the following Bellman equation:

$$\begin{aligned} V_t(k, \mathbf{M}, \mathbf{T}, J) = {} & \max_{c,\mu} u(c_t, l_t) + \frac{\beta}{1-\psi} \times \\ & \left[\mathbb{E}\left\{ \left((1-\psi)\, V_{t+1}\left(k^{+}, \mathbf{M}^{+}, \mathbf{T}^{+}, J^{+}\right)\right)^{\frac{1-\gamma}{1-\psi}} \right\} \right]^{\frac{1-\psi}{1-\gamma}}, \\ \text{s.t.} \quad & k^{+} = (1-\delta)k_t + \mathcal{Y}_t(k, T^{\mathrm{AT}}, \mu, J) - c_t, \\ & \mathbf{M}^{+} = \mathbf{\Phi}^{\mathrm{M}}\mathbf{M} + \left(\mathcal{E}_t\left(k, \mu\right), 0, 0\right)^{\top}, \\ & \mathbf{T}^{+} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{T} + \left(\xi_1 \mathcal{F}_t\left(M^{\mathrm{AT}}\right), 0\right)^{\top}, \\ & J^{+} = g^{J}(J, \mathbf{T}, \omega^{J}), \end{aligned} \tag{20}$$

for $t = 0, 1, \ldots, 599$. The terminal value function V_{600} is the terminal value function given in Cai et al. (2012a).

8.3 Numerical Examples

We implement our numerical dynamic programming to analyze how the optimal carbon tax is affected by different preference parameters combined with various tipping point events. We first solve a benchmark case with standard parameter assumptions. We cover a broad range of values for the degree of risk aversion γ and the reciprocal of the intertemporal elasticity of substitution ψ, representing empirical work that aims to estimate these

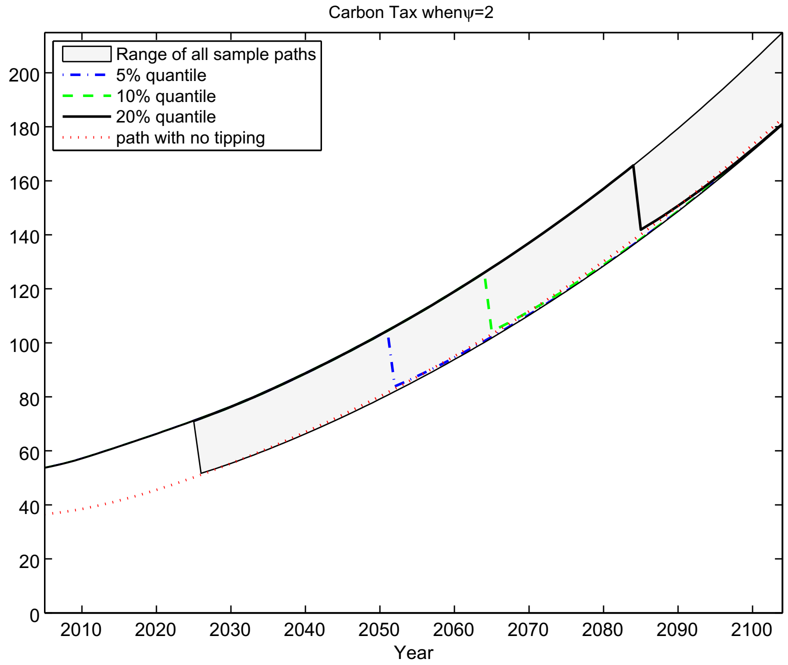


Figure 5 Carbon tax in the benchmark scenario.

parameters. In this section, we present the qualitative results of the benchmark case. The reader is referred to Cai et al. (2013a) for the specific parameter values and results of other cases.

Figure 5 illustrates the typical dynamic path for the carbon tax. The lower dotted line represents the optimal carbon tax if no tipping were expected. The upper envelope in Figure 5 represents at each time *t*, the carbon tax if there has not yet been a tipping event. We call this the pre-tipping carbon tax. In contrast, the lower envelope in general represents the carbon tax in the post-tipping regime. Figure 5 also displays the timing of some sample tipping events. For example, the first drop (which is at year 2025 in Figure 5) is the first tipping out of our 1000 simulations. By the middle of this century about 5% of the simulated paths have generated a tipping point and by the end of the 21st century more than 20% of the paths have exhibited a tipping point.

The key fact to note in Figure 5 is the effect of the possible tipping event on the carbon tax. At the initial period (year 2005), the optimal carbon tax is $54, while it is $37 in the case when the tipping point does not exist, a significant increase. Furthermore, note that the extra carbon tax due to the anticipated tipping event is roughly constant. Figure 5 illustrates an unexpected decomposition of the optimal carbon tax: the deterministic component of damages implies a growing carbon tax in the optimal policy, but the stochastic damages caused by the uncertain time of tipping imply a near constant increase in the optimal carbon tax.

Different assumptions about tastes, technology, and climate will produce different optimal carbon tax paths. However, the example in Figure 5 as well as many others in Cai et al. (2012a) clearly show that the stochastic component must be modeled explicitly via stochastic dynamic programming. No certainty equivalent can make the distinction between the sure damages and the uncertain damages. This is a clear example of where multidimensional dynamic programming is essential to arrive at an important insight regarding greenhouse gas policy.

9. CONCLUSIONS

Dynamic economic problems are analyzed with dynamic programming methods. Solving the complex economic multidimensional problems that economists study requires reliable and efficient algorithms. This chapter has reviewed a few of the key elements of any such effort, but recognizes that there is much left to be done regarding both mathematical methods and efficient utilization of available hardware.

For example, further development of multidimensional approximation methods is needed. Using sparse grid methods (Malin et al. (2011) contains an informative discussion of Smolyak's method, a sparse grid method used in economics) would dramatically reduce the number of points we use in our state space, but perhaps at the cost of not preserving shape. Numerical dynamic programming presents novel mathematical challenges due to the multiple objectives of using few points, computing an approximation that has small errors in a traditional metric such as $\mathcal{L}^\infty$, as well as computing an approximation that satisfies qualitative criteria such as monotonicity and curvature constraints. We suspect that algorithms that find efficient solutions to such approximation problems will themselves be computationally intensive, but in developing those methods we should remember that intensive computations will be worth the effort if the resulting value functions are evaluated thousands, or even millions, of times in later iterations.

By its definition, computational methods depend on the properties of computers, the hardware. As hardware changes, we will want to exploit novel features and capabilities. For example, the newest innovation in basic hardware is the exploding use of graphical processing units (GPUs) and the potential partnerships between GPUs and conventional CPUs in supercomputers. Hardware innovations are an essential part of the evolution of computational methods, and those developing methods will often find that new hardware alters the tradeoffs that must be balanced when creating numerical methods.

Hopefully this chapter will suffer the same fate as many other such chapters reviewing the state-of-the-art in computational tools and soon be viewed as being primitive and hopelessly out of date. Such is the nature of computational science.

ACKNOWLEDGMENTS

Cai and Judd gratefully acknowledge NSF support (SES-0951576), and thank Karl Schmedders and three anonymous referees for their helpful comments.

REFERENCES

Abrams, R.A., Karmarkar, U.S., 1980. Optimal multiperiod investment-consumption policies. Econometrica 48 (2), 333–353.

Bellman, R., 1957. Dynamic Programming. Princeton University Press.

Boyle, P.P., Lin, X., 1997. Optimal portfolio selection with transaction costs. North American Actuarial Journal 1 (2), 27–39.

Brown, D.B., Smith, J.E., 2011. Dynamic portfolio optimization with transaction costs: heuristics and dual bounds. Management Science 57 (10), 1752–1770.

Cai, Y., 2010. Dynamic Programming and Its Application in Economics and Finance. PhD Thesis, Stanford University.

Cai, Y., Judd, K.L., 2010. Stable and efficient computational methods for dynamic programming. Journal of the European Economic Association 8 (2–3), 626–634.

Cai, Y., Judd, K.L., 2012a. Dynamic programming with shape-preserving rational spline Hermite interpolation. Economics Letters 117 (1), 161–164.

Cai, Y., Judd, K.L., 2012b. Dynamic programming with Hermite approximation. NBER Working Paper No. 18540.

Cai, Y., Judd, K.L., 2013. Shape-preserving dynamic programming. Mathematical Methods of Operations Research 77 (3), 407–421.

Cai, Y., Judd, K.L., Lontzek, T.S., 2012a. Continuous-time methods for integrated assessment models. NBER Working Paper No. 18365.

Cai, Y., Judd, K.L., Lontzek, T.S., 2012b. DSICE: a dynamic stochastic integrated model of climate and economy. RDCEP Working Paper No. 12-02.

Cai, Y., Judd, K.L., Lontzek, T.S., 2013a. The social cost of stochastic and irreversible climate change. NBER Working Paper No. 18704.

Cai, Y., Judd, K.L., Thain, G., Wright, S., 2013b. Solving dynamic programming problems on a computational grid. NBER Working Paper No. 18714.

Cai, Y., Judd, K.L., Xu, R., 2013c. Numerical solution of dynamic portfolio optimization with transaction costs. NBER Working Paper No. 18709.

Constantinides, G.M., 1976. Optimal portfolio revision with proportional transaction costs: extension to HARA utility functions and exogenous deterministic income. Management Science 22 (8), 921–923.

Constantinides, G.M., 1979. Multiperiod consumption and investment behavior with convex transaction costs. Management Science 25, 1127–1137.

Constantinides, G.M., 1986. Capital market equilibrium with transaction costs. Journal of Political Economy 94 (4), 842–862.

Costantini, P., Fontanella, F., 1990. Shape-preserving bivariate interpolation. SIAM Journal of Numerical Analysis 27, 488–506.

Czyzyk, J., Mesnier, M., Moré, J., 1998. The NEOS Server. IEEE Journal on Computational Science and Engineering 5, 68–75.

Den Haan, W.J., Judd, K.L., Juillard, M., 2011. Computational suite of models with heterogeneous agents II: multi-country real business cycle models. Journal of Economic Dynamics & Control 35, 175–177.

Epstein, L.G., Zin, S.E., 1989. Substitution, risk aversion, and the temporal behavior of consumption and asset returns: a theoretical framework. Econometrica 57 (4), 937–969.

Fourer, R., Gay, D.M., Kernighan, B.W., 1990. Modeling language for mathematical programming. Management Science 36, 519–554.

Gennotte, G., Jung, A., 1994. Investment strategies under transaction costs: the finite horizon case. Management Science 40 (3), 385–404.

Gill, P., Murray, W., Saunders, M.A., Wright, M.H., 1994. User's Guide for NPSOL 5.0: a Fortran Package for Nonlinear Programming. Technical Report, SOL, Stanford University.

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

Goodman, T.N.T., 2001. Shape preserving interpolation by curves. In: Proceedings of the 2001 International Symposium, pp. 24–35.

Griebel, M., Wozniakowski, H., 2006. On the optimal convergence rate of universal and nonuniversal algorithms for multivariate integration and approximation. Mathematics of Computation 75 (255), 1259–1286.

Interagency Working Group on Social Cost of Carbon, 2010. Social Cost of Carbon for Regulatory Impact Analysis under Executive Order 12866. United States Government. <http://www.whitehouse.gov/sites/default/files/omb/inforeg/for-agencies/Social-Cost-of-Carbon-for-RIA.pdf>.

Judd, K.L., 1998. Numerical Methods in Economics. The MIT Press.

Judd, K.L., Solnick, A., 1994. Numerical dynamic programming with shape-preserving splines. Hoover Institution.

Judd, K.L., Kubler, F., Schmedders, K., 2012. Bond ladders and optimal portfolios. Review of Financial Studies 24 (12), 4123–4166.

Juillard, M., Villemot, S., 2011. Multi-country real business cycle models: accuracy tests and test bench. Journal of Economic Dynamics & Control 35, 178–185.

Kamin, J.H., 1975. Optimal portfolio revision with a proportional transaction cost. Management Science 21 (11), 1263–1271.

Kreps, D.M., Porteus, E.L., 1978. Temporal resolution of uncertainty and dynamic choice theory. Econometrica 46 (1), 185–200.

Lenton, T.M., Held, H., Kriegler, E., Hall, J., Lucht, W., Rahmstorf, S., Schellnhuber, H.J., 2008. Tipping elements in the Earth's climate system. PNAS 105, 1786–1793.

Lontzek, T.S., Cai, Y., Judd, K.L., 2012. Tipping Points in a Dynamic Stochastic IAM. RDCEP Working Paper No. 12-03.

Malin, B.A., Krueger, D., Kubler, F., 2011. Solving the multi-country real business cycle model using a Smolyak collocation method. Journal of Economic Dynamics & Control 35, 229–239.

Mehra, R., Prescott, E.C., 1985. The equity premium: a puzzle. Journal of Monetary Economics 15 (2), 145–161.

Murtagh, B., Saunders, M., 1982. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. Mathematical Programming Study 16, 84–117.

Nordhaus, W.D., 2008. A Question of Balance: Weighing the Options on Global Warming Policies. Yale University Press.

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, second ed. Palgrave Macmillan.

Rust, J., Traub, J.F., 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 Journal of Numerical Analysis 20, 854–864.

Stachurski, J., 2008. Continuous state dynamic programming via nonexpansive approximation. Computational Economics 31, 141–160.

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

Zabel, E., 1973. Consumer choice, portfolio decisions, and transaction costs. Econometrica 41 (2), 321–335.