SPLINE APPROXIMATION OF POLICY FUNCTIONS IN ECONOMIC DYNAMICS WITH UNCERTAINTIES

MOODY T. CHU*, CHUN-HUNG KUO†, AND MATTHEW M. LIN‡

Abstract. Modern economic theory views the economy as a dynamical system in which rational decisions are made in the face of uncertainties. The dynamics includes changes over time of market behavior such as consumption, investment, labor supply, and technology innovation, all interpreted in a broad sense. The Euler equation arises as the first order optimality condition when solving an economic dynamics system. Finding the policy function inherent in the Euler equation is an important but challenging task. This note proposes a Newton iterative scheme on approximating the unknown policy functions by composite 1-dimensional cubic splines. This spline approach has the advantages of freedom in the node collocation, simplicity in the derivative calculation, fast convergence, and high precision over the conventional projection methods. Applications to the neoclassical growth model with leisure choice are used to demonstrate the working of the idea. In particular, tensor products are employed to simplified and effectuate the operations.

Key words. economic dynamics, dynamical programming, Bellman equation, Euler equation, policy function, cubic spline, tensor operation

AMS subject classifications. 37B35, 37N40, 90C22, 90C51

1. Introduction. To sketch a quick background of the subject, we start with a simple model. Given initial values of capital k_0 and technology z_0, the social planner needs to make optimal decision on the sequences of consumptions $\{c_t\}_{t=0}^{\infty}$ and capital accumulations $\{k_{t+1}\}_{t=0}^{\infty}$ throughout the time so as to [1]

$$\begin{aligned} \max_{\{c_t,k_{t+1}\}_{t=0}^{\infty}} \quad & \mathcal{E}_0\left[\textstyle\sum_{t=0}^{\infty}\beta^t u(c_t)\right], \\ \text{subject to} \quad & c_t + k_{t+1} = f(k_t, z_t), \\ & z_{t+1} = \rho z_t + \epsilon_{t+1}. \end{aligned} \tag{1.1}$$

In the constraints, $f(k_t, z_t)$ denotes the total production of the economy at the time period t and is determined by the current capital k_t and technology level z_t. The essence of uncertainty in this setting is caused by the evolution of the technology z_t which is assumed to follow a first order autoregressive process with normal innovation $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. In the objective function, the utility function $u(c_t)$ simulates in a broad sense the "level of satisfaction" that can be brought forth by the consumption c_t while the discount rate β characterizes the inclination of preferring consumption today than tomorrow. Both functions u and f are predetermined and their desirable properties will be described in the subsequent discussion. The operator $\mathcal{E}_0$, with a noticeable subscript $_0$, emphasizes that the expected value is taken over technologies $\{z_t\}_{t=1}^{\infty}$ conditioned upon the initial technology z_0. The optimization problem (1.1) usually is referred to as *the social planner's problem*. The resulting optimal value of (1.1), as a function of the initial state of capital k_0 and technology z_0, is referred to as *the value function* and is denoted by $v(k_0, z_0)$.

Before continuing, we caution readers of the indicative but somewhat inadvertent misnomer that the subscript $_t$, commonly adopted in the economics literature, refers exclusively to the time period t of the economy evolution and should not be confused with the subscript $_i$ customarily used in the mathematics literature as a pointer or entry index of an array.

*Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205. (chu@math.ncsu.edu). This research was supported in part by the National Science Foundation under grants DMS-0732299 and DMS-1014666.

†Department of Economics, North Carolina State University, Raleigh, NC 27698-8109. (ckuo2@unity.ncsu.edu).

‡Department of Mathematics, National Chung Cheng University, Chiaya 62012 Taiwan. (mlin@math.ccu.edu.tw). This research was supported in part by the National Science Council of Taiwan under grant 99-2115-M-194-010-MY2.

Bellman's principle of optimality. Problem (1.1) involves infinitely many decisions simultaneously, which makes the solution extremely difficult to find. What is even more challenging is that solving (1.1) necessarily means to find the solution sequences $\{c_t\}_{t=0}^{\infty}$ and $\{k_{t+1}\}_{t=0}^{\infty}$ for every given initial state (k_0, z_0) because, whenever the initial state is changed, so are the subsequent decisions. One ingenious insight of great importance by Bellman reformulates the problem in a recursive form which significantly reduces the computation complexity [16]. The idea, known as Bellman's principle of optimality, asserts that an optimal policy, if exists, should have the property that the subsequent decisions from any given initial state and decision remain optimal with regard to the state resulting from the first decision [3]. In other words, the value function $v(k_t, z_t)$ should satisfy *the Bellman equation*,

$$v(k_t, z_t) = \max_{c_t, k_{t+1}} \left\{ u(c_t) + \beta \mathcal{E}_t \left[v(k_{t+1}, z_{t+1}) \right] \right\}, \tag{1.2}$$

subject to the same constraints as in (1.1). Note that only two variables, linearly dependent due to the resource constraint, are involved in the maximization of (1.2).

The Bellman equation (1.2) typifies a general dynamic programming problem which arises in many areas other than economics [4, 6]. Quite a few numerical methods have already been proposed in the literature for solving (1.2). See, for example, a careful comparison of eight different algorithms developed from notions of either perturbation or projection in [2] and general discussions in [9, 11]. The emphasis of this paper is to propose a method that juxtapose the freedom of node collocation and the simplicity of projection without using basis.

Euler equation. Though it appears often that the value function $v(k_t, z_t)$ is the underlying unknown in (1.2), for the purpose of decision-making it is sometimes more desirable to obtain the *policy function*

$$k_{t+1} = p(k_t, z_t) \tag{1.3}$$

which describes the economy agent's optimal behavior with respect to state variables k_t and z_t. Repeated applications of the policy function induce the dynamics in the sequential decisions. Toward this end, we formulate the Lagrange function

$$L(c_t, k_{t+1}, z_t; \lambda_t) := u(c_t) + \beta \mathcal{E}_t \left[v(k_{t+1}, z_{t+1}) \right] - \lambda_t (c_t + k_{t+1} - f(k_t, z_t)) \tag{1.4}$$

with λ_t as the multiplier. The first order optimality condition requires that

$$\begin{cases} \frac{\partial L}{\partial c_t} &= \frac{du(c_t)}{dc_t} - \lambda_t = 0, \\ \frac{\partial L}{\partial k_{t+1}} &= \beta \mathcal{E}_t \left[\frac{\partial v(k_{t+1}, z_{t+1})}{\partial k_{t+1}} \right] - \lambda_t = 0, \\ \frac{\partial L}{\partial \lambda_t} &= c_t + k_{t+1} - f(k_t, z_t) = 0. \end{cases} \tag{1.5}$$

Applying the envelope theorem to the Bellman equation, we see that

$$\frac{\partial v(k_t, z_t)}{\partial k_t} = \frac{\partial L}{\partial k_t} = \lambda_t \frac{\partial f(k_t, z_t)}{\partial k_t}. \tag{1.6}$$

By eliminating the multiplier λ_t in the first two equations of (1.5), it follows that a necessary condition for optimality is

$$\frac{du(c_t)}{dc_t} = \beta \mathcal{E}_t \left[\frac{du(c_{t+1})}{dc_{t+1}} \frac{\partial f(k_{t+1}, z_{t+1})}{\partial k_{t+1}} \right], \tag{1.7}$$

which is known as *the Euler equation* for the system (1.2). We can replace c_t and c_{t+1} in (1.7) by the relationship

$$c_t = f(k_t, z_t) - k_{t+1},$$

which, after writing

$$\Gamma(k_t, k_{t+1}, z_t) := \frac{du(f(k_t, z_t) - k_{t+1})}{dc_t},$$
$$\Xi(k_t, z_t) := \frac{\partial f(k_t, z_t)}{\partial k_t},$$

leads to a 3-term finite difference equation

$$\Gamma(k_t, k_{t+1}, z_t) = \beta \mathcal{E}_t \left[\Gamma(k_{t+1}, k_{t+2}; z_{t+1}) \Xi(k_{t+1}, z_{t+1})\right] \tag{1.8}$$

for the unknowns $\{k_t, k_{t+1}, k_{t+2}\}$.

One point should be made clear. Starting with a given initial k_0 and an arbitrary k_1, it seems natural that the sequence $\{k_t\}$ generated by solving (1.8) would automatically satisfy the corresponding Euler equation. This is a misconception, however. Merely having a sequence of iterates satisfying (1.8) is not enough. The trouble is that the curve or, more precisely, the surface that interpolates these iterates may not satisfy the Euler equation in its entirety due to an incorrect value of k_1 which is supposedly equal to the evaluation of the unknown policy function $p(k_0, z_0)$, as is illustrated in Figure 1.1 for a fixed z_0.

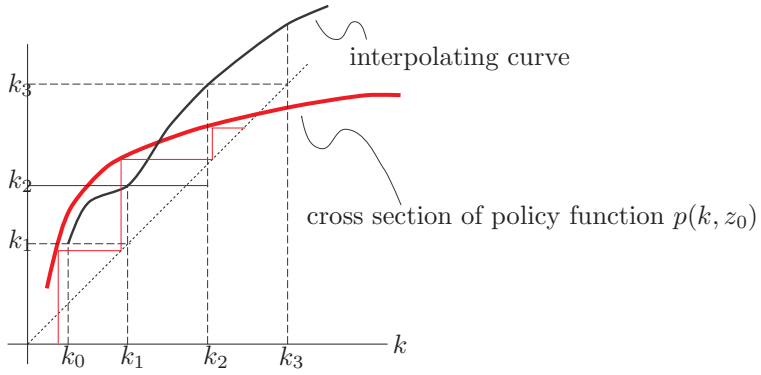

FIG. 1.1. *Interpolating curve at discrete points obtained from (1.8) versus true policy function* $p(k, z_0)$.

What we are interested in is to find the policy function (1.3),which is a 2-term relationship, so as to satisfy

$$\frac{du(f(k_t, z_t) - p(k_t, z_t))}{dc_t} = \beta \mathcal{E}_t \left[\frac{du\left(f(p(k_t, z_t); z_{t+1}) - p(p(k_t, z_t), z_{t+1})\right)}{dc_{t+1}} \frac{\partial f(p(k_t, z_t); z_{t+1})}{\partial k_{t+1}}\right] \tag{1.9}$$

for any given (k_t, z_t). Since both u and f are specified a priori, we see upon integrating the right side of (1.7) over z_{t+1} for the expected value that the Euler equation in general is a deterministic functional equation

$$F(k_t, z_t, p(k_t, z_t)) = 0 \tag{1.10}$$

for the unknown function $p(k_t, z_t)$, where $F : \mathbb{R}^3 \to \mathbb{R}$ is some known nonlinear function. We shall be more specific about the constituents of the function F in the subsequent discussion, but included in the functional equation (1.10) for the problem (1.1) is the interesting but challenging task of reducing the three-term recurrence relation (1.8) to a two-term relation (1.3).

Conditions for term reduction. Obviously, conventional wisdom infers that the notion of reducing a three-term relationship to a two-term is impossible in general. Existence can be assumed only under some special circumstances. The following "standard" assumptions,

- $0 < \beta < 1$;
- The utility function u is continuously differentiable, strictly concave, and strictly increasing;
- The production function $f(\cdot, z_t)$ is continuous differentiable, concave, strictly increasing, $f(0, \cdot) \equiv 0$, and there is $\bar{k} > 0$ such that

$$f(k, z_t) \begin{cases} > k & \text{if } k < \bar{k}, \\ < k & \text{if } k > \bar{k}; \end{cases}$$

acquired after extensive observations and explorations by economists, seem adequate to characterize general economic dynamics well. Most importantly, it is now a classical result that under these assumptions the value function and the corresponding policy function for the model problem (1.1) exist and are unique [16].

Of course, there are many other more complicated models taking into account other factors for economic dynamics, but the central theme is about we have just described — finding either the value function or the policy function. In this paper, our goal is to outline how the notion of 1-dimensional spline approximation can readily be applied to economic dynamics. Note that we accentuate *the usage of 1-dimensional spline only* of which we gain many advantages. For demonstration purpose, we shall limit our attention to the neoclassical growth model with leisure choice,

$$\begin{aligned} \max_{\{c_t, k_{t+1}, \ell_t\}_{t=0}^{\infty}} \quad & \mathcal{E}\left[\sum_{t=0}^{\infty} \beta^t \frac{\left(c_t^{\theta}(1-\ell_t)^{1-\theta}\right)^{1-\eta}}{1-\eta}\right], \\ \text{subject to} \quad & c_t + k_{t+1} = e^{z_t} k_t^{\alpha} \ell_t^{1-\alpha} + (1-\delta) k_t, \\ & z_{t+1} = \rho z_t + \epsilon_{t+1}, \end{aligned} \tag{1.11}$$

where $0 \le \ell_t \le 1$ stands for the labor supply at time t and, hence, $1 - \ell_t$ denotes the leisure. This additional variable makes (1.11) more complicated than (1.1), yet our idea remains generalizable. Our emphasis is on the simplicity and sufficiency of only a few break points needed in the spline application for high dimensional and high resolution approximation. Additionally, we exploit the succinct programming style by means of tensor operations[1].

2. Spline approximation. Many options are available for approximating the policy function $p(k, z)$. See, for example, the discussion in [2, 11, 16]. We propose to approximate the policy function for the model (1.11) by a 2-dimensional cubic spline and solve the resulting system of discrete Euler equation by the Newton method. Using spline approximation and recursive methods certainly is not a new idea. A comprehensive discussion on this subject can be found in the seminal book [16]. See also [7, 12, 14, 15] for applications of shape-preserving splines to dynamical programming. Our approach stands out, however, as a special 2-dimensional spline which really can be thought of as one 1-dimensional spline "weighted" by another 1-dimensional spline. Since our formulation is essentially a 1-dimensional spline, it allows us to take advantage of the easy calculation of its derivatives analytically as we shall see in the subsequent discussion.

Recall that splines are local interpolations with controlled behavior — slope, curvature, and other degrees of differentiability — at places where two local interpolating pieces meet. We mention the cubic spline as a possible interpolant only for its ease to use. The more general concept of B-spline [8] could also be used which, in particular, offers more control over the differentiability of the spline at points where the policy function displays "kinks". Because of space limitation, we choose not to explore this generalization in this presentation.

It might be instructive to first explain how the data are structured in the Matlab environment. We then show that, even though the analytic form of the spline might be hidden from sight, we can

[1]The tensor operations we intend to exploit in this paper are different from the notion of n-fold tensor product mentioned in the seminal book [9, Sections 4.2.1 and 8.2.5] for multiple variable product base for projection methods. Our implementations do, for example, highly optimized matrix to matrix multiplications.

calculate its derivatives, especially its sensitivity to interpolants, point by point up to the machine precision. Thus we may take the full advantage of quadratic convergence of the Newton method for computing the policy function.

Representing a cubic spline. Given $\{(x_i, y_i)\}_{i=1}^n$, the cubic spline $q(k)$ that interpolates these points is a piecewise function of the form that for $i = 1, \ldots, n-1$,

$$q(k) = y_i + b_i(k - x_i) + c_i(k - x_i)^2 + d_i(k - x_i)^3, \quad k \in [x_i, x_{i+1}]. \tag{2.1}$$

Let $\mathbf{x} := [x_1, \ldots, x_n]$ and $\mathbf{y} := [y_1, \ldots, y_n]$. With appropriate boundary constraints[2], the Matlab command

```
A = spline(x,y);
```

creates a *structure field* of the form

```
A =
      form: 'pp'
    breaks: [1xn double]
     coefs: [(n-1)x4 double]
    pieces: n-1
     order: 4
       dim: 1
```

where `n` is actually the numeric n of the length of the `breaks` $x_1, \ldots, x_n$ and `coefs` is an $(n-1) \times 4$ matrix, retrievable from the command `A.coefs`, that stores the coefficients for the spline,

$$\texttt{A.coefs} = \begin{bmatrix} d_1 & c_1 & b_1 & y_1 \\ \vdots & \vdots & & \\ d_{n-1} & c_{n-1} & b_{n-1} & y_{n-1} \end{bmatrix}.$$

As the structure field `A` contains the essential information of the spline, it can be passed into the Matlab command such as

```
qk = ppval(A,k);
```

which returns the evaluation $q(k)$ at any desirable point (or array of points) k.

Derivatives of a cubic spline. For our applications, we need to compute two kinds of derivatives of a spline. First, we need the "ordinary" derivative of $q(k)$ evaluated at y_j. Because

$$\frac{dq}{dk} = b_i + 2c_i(k - x_i) + 3d_i(k - x_i)^2$$

over the interval $[x_i, x_{i+1}]$, we may characterize the piecewise polynomial $\frac{dq}{dk}$ by the structure field `dA` which has the same structure as `A` except that its `dA.coefs` is modified to

```
dA.coefs(:,1) = zeros(size(A.coefs,1),1);
dA.coefs(:,2) = 3*A.coefs(:,1);
dA.coefs(:,3) = 2*A.coefs(:,2);
dA.coefs(:,4) = A.coefs(:,3);
```

[2]All demonstrations in this paper employ the not-a-knot boundary condition, that is, enforcing third derivative continuity across the second and penultimate knots of the spline, which is the default setting in Matlab.

and `ppval(dA,k)` evaluates the derivative at any given k. Next we need the sensitivity matrix of the spline to its parameters, i.e., the partial derivatives of the spline $q(k; y_1, \ldots y_n)$ with respect to each y_j, $j = 1, \ldots, n$. While it is known that the function `spline(x,y)` responses nonlinearly to changes in `x`, we argue that its response to changes in `y` is easy to compute. The fact comes from the realization that the coefficients (b_i, c_i, d_i) of the various cubic polynomials in the interpolating spline are entries of the solution vector to a specific tridiagonal linear system of which the square matrix on the left side of the equation is made of entries such as $x_{i+1} - x_i$ and its powers whereas the vector on the right side is made of $y_{i+1} - y_i$ and its likes, but no powers. In other words, with fixed break points $\{x_1, \ldots, x_n\}$, the spline $q(k; y_1, \ldots, y_n)$ depends *linearly* on $\{y_1, \ldots, y_n\}$ [13]. It follows that the partial derivatives are splines themselves. More specifically, for $j = 1, \ldots n$, the partial derivative $\frac{\partial q}{\partial y_j}$ is precisely the spline that interpolates the data $\{(\mathbf{x}, \mathbf{e}_j)\}$, where $\mathbf{e}_j$ is the jth standard unit vector. Taking advantage of the vector operations in Matlab, a simple one-line command

```
D = spline(x,eye(n));
```

where `eye(n)` refers to the identity matrix of size $n \times n$, effectively generates the matrix `D.coefs` that has $n - 1$ blocks of size $n \times 4$ where the jth row in the ith block stores the coefficients of the spline that interpolates $\mathbf{e}_j$ over the interval $[x_i, x_{i+1}]$. The evaluation

```
ppval(D,y)';
```

yields the $n \times n$ matrix $\left[\frac{\partial q(y_i)}{\partial y_j}\right]$

2-dimensional spline. The notion of splines can be generalized to higher dimensions. See, for example, [8, 15]. One such a generalization is the so called bicubic spline which has the advantage of being straightforward and guarantees continuity of only gradient and cross-derivative. Its second derivatives however, could be discontinuous. Instead, we propose a modified cubic spline approximation as follows.

Suppose the surface

$$z = h(x, y)$$

over the domain $\Omega \subset \mathbb{R}^2$ is to be approximated. Let $\mathbf{y} = [y_1, \ldots, y_n]$ be a preselected set of feasible z values and define

```
W = spline(y,eye(n));
```

or

```
W = spline(y,[zeros(n,1),eye(n),zeros(n,1)]);
```

with zero end slopes clamped splines. Contained in `W` are n splines $W_j(y)$, $j = 1, \ldots, n$, satisfying $W_j(y_i) = \delta_{ij}$. For each j, choose breakpoints $\mathbf{x}_j = [x_{j1}, \ldots x_{jd_j}]$ so that $(x_{js}, y_j) \in \Omega$ for all s and j and define $\mathbf{h}_j := [h(x_{j1}, y_j), \ldots, h(x_{jd_j}, y_j)]$. Note that $\mathbf{x}_i$ need not be the same as $\mathbf{x}_j$, nor have the same dimensionality, for different i and j. Compute the cubic spline

```
L_j = spline(x_j,f_j);
```

where `x_j` $= \mathbf{x}_j$ and `f_j` $= \mathbf{f}_j$, $j = 1, \ldots, n$, and define the function

$$L_j(x) := \begin{cases} \texttt{ppval(L\_j,x)}, & \text{if } \mathtt{l}_j \le x \le \mathtt{u}_j, \\ 0 & \text{otherwise,} \end{cases} \tag{2.2}$$

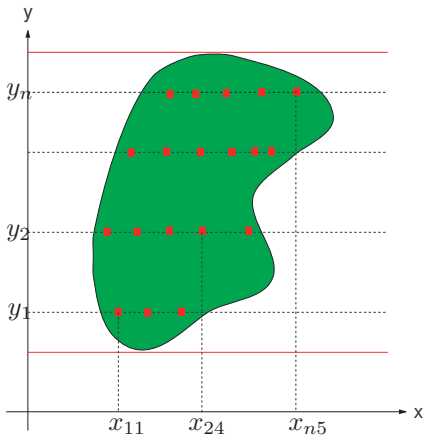

FIG. 2.1. *Breakpoints selection over the domain* Ω

where $[\mathfrak{l}_j, \mathfrak{u}_j]$ stands for the interval of cross section of the line $y = y_j$ and the domain Ω. We then define the bivariate function

$$S(x, y) := \sum_{j=1}^{n} L_j(x) W_j(y), \quad (x, y) \in \Omega. \tag{2.3}$$

It follows that for all $j = 1, \ldots, n$ and $s = 1, \ldots, d_j$ we have

$$S(x_{js}, y_j) = h(x_{js}, y_j).$$

For points that are not on the preselected grid, the functions $W_j(y)$, $J = 1, \ldots, n$, collectively play the role of "weighting" because for each y we have

$$\sum_{j=1}^{n} W_j(y) = 1,$$

though some of the weights might be negative. For our application, we are mainly interested in the case of rectangular domain Ω with same $\mathbf{x}_i$ for all i.

A Matlab demonstration. Consider the peaks function

$$f(x, y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{e^{-(x+1)^2-y^2}}{3}, \quad (x, y) \in [-3, 3] \times [-3, 3].$$

The following few lines of coding quickly constructs our weighted 1-D spline approximation.

```
N = 20;
[x,y,z] = peaks(N);                                  % generate surface test data

[xi,yi] = meshgrid(-3:.1:3,-3:.1:3);
v = peaks(xi,yi);                                    % exact surface

A = spline(x(1,:),z);                                % Generate 1D splines over x
B = spline(y(:,1),[eye(N)]);                         % Generate weights over y

w = ppval(B,yi(:,1))'*ppval(A,xi(1,:));              % Weighted 1D spline
```

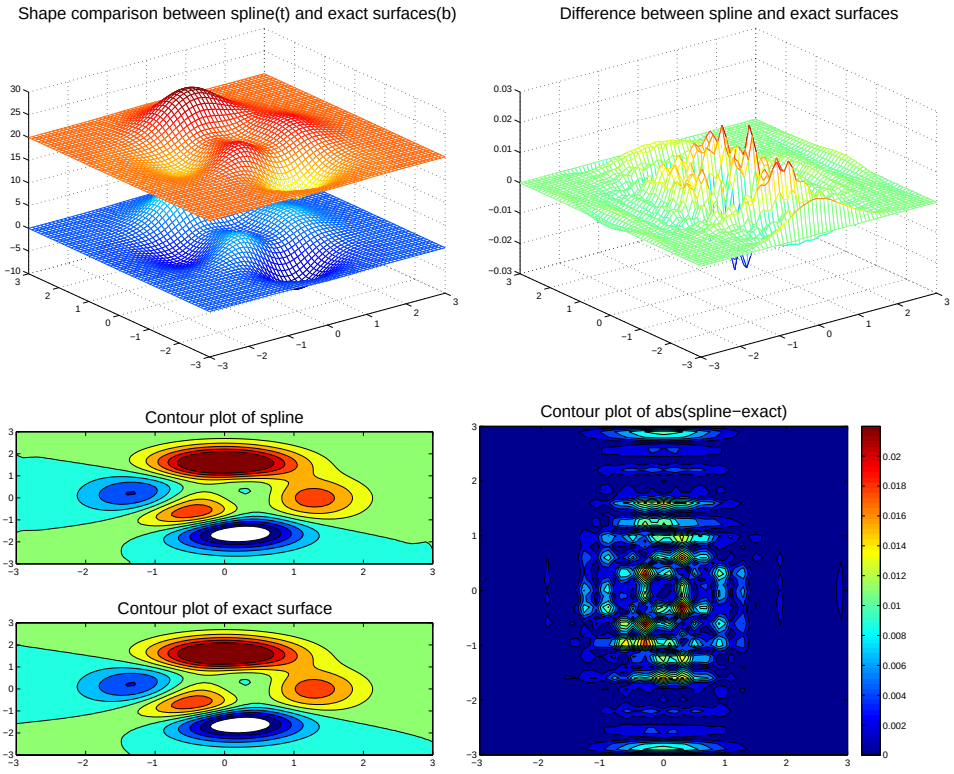

FIG. 2.2. *Comparison between approximate surface by (2.3) and the exact surface.*

We compare the approximate surface with the exact surface in Figure 2.2. In the upper left drawing, we elevate the spline surface by 20 units to show the respective terrains of the surfaces. The contour plots at the lower left drawing suggest a fairly close match of the two surfaces. The actual difference with the maximal absolute error of the order 10^{-2} is shown in the two drawings on the right column of Figure 2.2.

3. Application to the neoclassical growth model with leisure choice. For the model (1.11), we need to determine multiply policy functions. We have already mentioned in (1.3) the policy function $k_{t+1} = p(k_t, z_t)$ for the next step of capital. We also need

$$\ell_t = r(k_t, z_t) \tag{3.1}$$

for the current step of labor. Be aware of the time difference. It can be shown that the Euler equation is given by

$$\begin{cases} \dfrac{\left(c_t^\theta (1-\ell_t)^{1-\theta}\right)^{1-\eta}}{c_t} &= \beta \mathcal{E}_t \left[\dfrac{\left(c_{t+1}^\theta (1-\ell_{t+1})^{1-\theta}\right)^{1-\eta}}{c_{t+1}} \left(\alpha e^{z_{t+1}} k_{t+1}^{\alpha-1} \ell_{t+1}^{1-\alpha} + 1 - \delta \right) \right], \\ (1-\theta) \dfrac{\left(c_t^\theta (1-\ell_t)^{1-\theta}\right)^{1-\eta}}{1-\ell_t} &= \theta \dfrac{\left(c_t^\theta (1-\ell_t)^{1-\theta}\right)^{1-\eta}}{c_t} (1-\alpha) e^{z_t} k_t^\alpha \ell_t^{-\alpha}, \\ c_t + k_{t+1} &= e^{z_t} k_t^\alpha \ell_t^{1-\alpha} + (1-\delta) k_t. \end{cases} \tag{3.2}$$

The last two equations in (3.2) are static in time and give rise to the explicit relationships

$$\begin{cases} c_t &= \frac{\theta(1-\alpha)}{1-\theta} e^{z_t} k_t^{\alpha} \ell_t^{-\alpha} (1-\ell_t), \\ k_{t+1} &= e^{z_t} k_t^{\alpha} \ell_t^{1-\alpha} + (1-\delta) k_t - \frac{\theta(1-\alpha)}{1-\theta} e^{z_t} k_t^{\alpha} \ell_t^{-\alpha} (1-\ell_t), \end{cases} \tag{3.3}$$

which implies that, once the policy function for labor supply ℓ_t is determined, then the policy functions for consumption c_t and next period capital k_{t+1} are also determined. Thus we only need to concentrate on finding the policy function for the labor supply. Substituting (3.3) into the first equation in (3.2), we obtain the expression

$$\begin{aligned} &\left(e^{z_t} k_t^{\alpha} \ell_t^{-\alpha}\right)^{\theta(1-\eta)-1} (1-\ell_t)^{-\eta} = \\ &\beta \mathcal{E}_t \left[\left(e^{z_{t+1}} \left(e^{z_t} k_t^{\alpha} \ell_t^{1-\alpha} + (1-\delta) k_t - \frac{\theta(1-\alpha)}{1-\theta} e^{z_t} k_t^{\alpha} \ell_t^{-\alpha} (1-\ell_t) \right)^{\alpha} \ell_{t+1}^{-\alpha} \right)^{\theta(1-\eta)-1} (1-\ell_{t+1})^{-\eta} \right. \\ &\left. \left(\alpha e^{z_{t+1}} \left(e^{z_t} k_t^{\alpha} \ell_t^{1-\alpha} + (1-\delta) k_t - \frac{\theta(1-\alpha)}{1-\theta} e^{z_t} k_t^{\alpha} \ell_t^{-\alpha} (1-\ell_t) \right)^{\alpha-1} \ell_{t+1}^{1-\alpha} + 1 - \delta \right) \right]. \end{aligned} \tag{3.4}$$

Based on the preceding section, the policy function (3.1) for labor supply is to be approximated by a 2-dimensional spline in the form of (2.3). We outline somewhat more programming details below. In particular, we demonstrate that the calculation can be done conveniently in terms of tensor product.

Labor supply and capital approximation. Suppose that the matrix $\mathtt{R} = R = [r_{ij}] \in \mathbb{R}^{m \times n}$ denotes the discrete approximation of the policy function (3.1) for labor supply at a prescribed grid $\{(x_i, y_j)\}$. The spline $S(x, y)$ is expected to satisfy the interpolation criteria

$$S(x_i, y_j) = r_{ij},$$

where the values of r_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, n$, are to be determined. Specifically, the simple command

```
L = spline(x,R');
```

alone defines n individual 1-dimensional splines where each individual spline L_s, $s = 1, \ldots, n$, can be characterized, if so desired, via

```
L_s = L;
L_s.dim = 1;
L_s.coefs = L.coefs(s:n:end,:);
```

and satisfies `ppval(L_s,x)` $= [r_{1s}, \ldots, r_{ms}]$. Consider each fixed (x_i, y_j) in the grid as the current capital k_t and technology z_t. The quantity

$$\mathfrak{X}_{ij} := e^{y_j} \left(\frac{x_i}{r_{ij}} \right)^{\alpha} r_{ij} + (1-\delta) x_i - \frac{\theta(1-\alpha)}{1-\theta} e^{y_j} \left(\frac{x_i}{r_{ij}} \right)^{\alpha} (1 - r_{ij})$$

then mimics the next period capital k_{t+1}, given the various values of current (k_t, z_t), A graphic rendition of these relationship is sketched in Figure 3.1, where the surfaces depicted there do not mean to be realistic.

Discrete Markov process. We must keep in mind that the next period labor supply $\ell_{t+1} = r(k_{t+1}, z_{t+1})$ is indeed stochastic because of its dependence on $z_{t+1} = \rho z_t + \epsilon_{t+1}$. There is a wonderful technique by Tauchen that approximates the stochastic process z_{t+1} by a finite Markov chain [17] and,

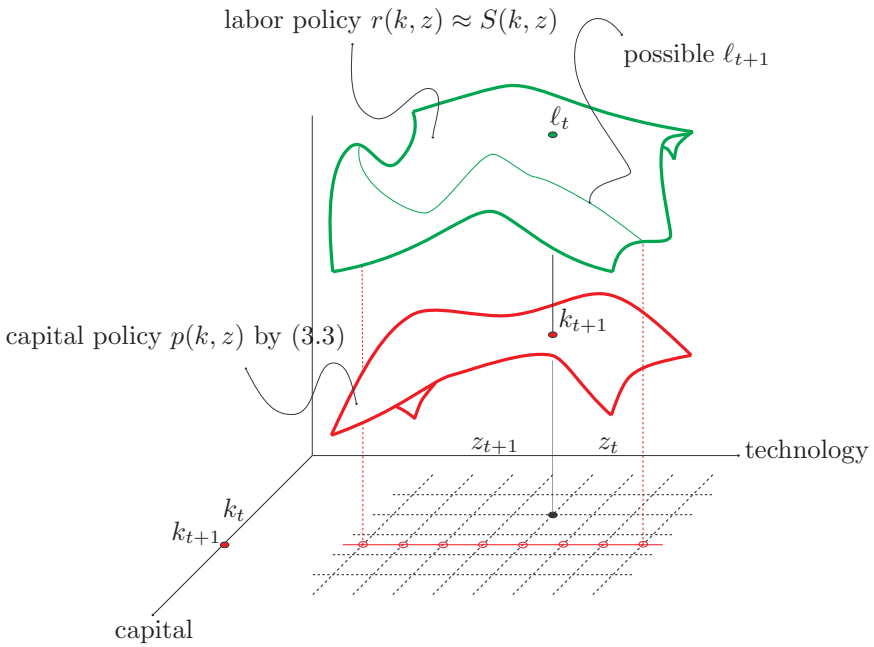

FIG. 3.1. *Sketch of policy functions $p(k,z)$ and $r(k,z)$ versus the (k,z) domain.*

hence, facilitates the computation of the expected value. An application of this method constructs finitely many possible states of technology $\{y_1, y_2, \ldots, y_n\}$, based on initial z_0, with the transition matrix $\Pi = [\pi_{js}]$ where π_{js} stands for the conditional probability

$$\pi_{js} := \text{Prob}\{z_{t+1} = y_s | z_t = y_j\}.$$

Then with the abbreviation

$$\zeta := \theta(1-\eta) - 1,$$

the conditional expected value in (3.4) for each given $z_t = y_j$ should be written as

$$\sum_{s=1}^{n} \pi_{js} \left(e^{y_s} \left(\frac{\mathfrak{X}_{ij}}{L_s(\mathfrak{X}_{ij})} \right)^{\alpha} \right)^{\zeta} (1 - L_s(\mathfrak{X}_{ij}))^{-\eta} \left(\alpha e^{y_s} \left(\frac{\mathfrak{X}_{ij}}{L_s(\mathfrak{X}_{ij})} \right)^{\alpha-1} + 1 - \delta \right).$$

Introduce the tensor

$$\mathfrak{R}_{sij} := L_s\left(\mathfrak{X}_{ij}\right) \tag{3.5}$$

representing the next period labor supply ℓ_{t+1} due to the policy function (3.1) evaluated at the next period technology z_{t+1} which is stochastic with random values z_s, $s = 1, \ldots, n$, and the next period capital $\mathfrak{X}_{ij}$ which is determined by the various values of current (k_t, z_t). By (3.4), we are interested in solving the nonlinear system

$$F(r_{11}, \ldots, r_{mn}) := \begin{bmatrix} F_{11} & F_{21} & \ldots & F_{n1} \\ F_{21} & F_{22} & & \\ \vdots & & & \\ F_{m1} & F_{m2} & \ldots & F_{mn} \end{bmatrix} = 0, \tag{3.6}$$

for the unknowns $\{r_{ij}\}$ so as to create the appropriate surface $S(x, y)$ for the labor supply policy function (3.1), where

$$F_{ij}(r_{11}, \ldots, r_{mn}) := \left(e^{y_j} \left(\frac{x_i}{r_{ij}} \right)^{\alpha} \right)^{\zeta} (1 - r_{ij})^{-\eta} - \beta \sum_{s=1}^{n} \pi_{js} \left(e^{y_s} \left(\frac{\mathfrak{X}_{ij}}{\mathfrak{R}_{sij}} \right)^{\alpha} \right)^{\zeta} (1 - \mathfrak{R}_{sij})^{-\eta} \left(\alpha e^{y_s} \left(\frac{\mathfrak{X}_{ij}}{\mathfrak{R}_{sij}} \right)^{\alpha - 1} + 1 - \delta \right).$$

Tensor operations. As most of the operations or evaluations involved are similar from element to element, it will be convenient to adopt MATLAB notations `.*`, `./`, and `.^` for element-to-element multiplication, division, or powers of matrices, respectively. In doing so, not only the task of programming is simplified, but more importantly the computation is effectively streamlined by exploiting fast matrix to matrix operations.

So as to describe the arithmetic operations appropriately in the MATLAB syntax, we define redundantly $\mathfrak{X} = \texttt{frakX} = [\mathfrak{X}_{ij}]$, $\mathfrak{R} = \texttt{frakR} = [\mathfrak{R}_{sij}]$, and so on, whereas all Greek letters are spelled out. Then, for example, the commands

```
recipR = 1 ./ R;
unoR = 1 - R;
Q = ((diag(x)*(recipR)).^alpha) * diag(exp(y));
frakX = Q.*R + (1-delta)*x'*ones(1,n) - theta*(1-alpha)/(1-theta)*Q.*unoR;

frakR = ppval(L,frakX);

F = zeros(m,n);
for s = 1:n
    tempR = squeeze(frakR(s,:,:));
    unotempR = 1 - tempR;
    frakQ = ((frakX./tempR).^alpha) * exp(y(s));
    partQ = (frakQ.*tempR) ./ frakX;
    F = F + ...
        (frakQ.^zeta .* unotempR.^(-eta) .* (alpha*partQ+1-delta)) * diag(Pi(:,s));
end

F = Q.^zeta .* unoR.^(-eta) - beta * F;
```

quickly construct the matrices $\mathfrak{X}$, $\mathfrak{R}$, and F. It might be informative to depict the above tensor product in Figure 3.2. The rectangular box represents the 3-dimensional tensor $\mathfrak{R}$. The way the entries of $\mathfrak{R}$ are stored is precisely in the order `frakR(s,i,j)`. So the horizontal slide in "green" color within this box represents the $m \times n$ matrix `tempR` for a fixed s. The tensor product we need at present for computing F is to scale columns of an $m \times n$ matrix associated with `tempR` by the diagonal matrix from the sth column of Π for $s = 1, \ldots, n$. This product effectively take care of the computation of expected values. Later, we shall use similar drawings to help to visualize multiplications under different meanings.

To compute the Jacobian matrix of $F(r_{11}, \ldots, r_{mn})$, we divide the construction process into several blocks. Firstly, we need the componentwise differentiation of a matrix with respect to R, defined by

$$\frac{\partial \mathfrak{X}}{\partial R} = \texttt{dfrakX.dR} := \left[\frac{\partial \mathfrak{X}_{ij}}{\partial r_{ij}} \right], \tag{3.7}$$

which can easily be calculated by the command

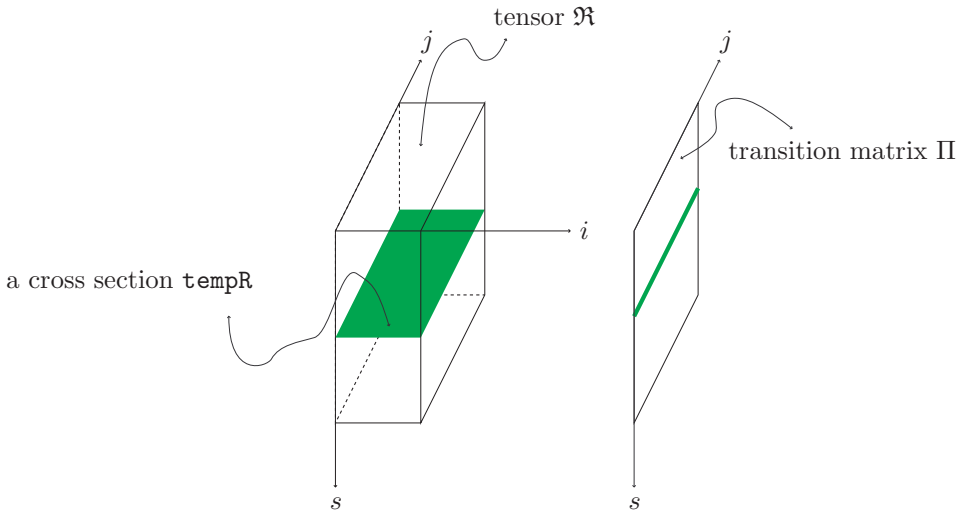

FIG. 3.2. *Diagram for tensor product*

```
dfrakX.dR =  (1-alpha)*Q.*(1+theta/(1-theta)*(1-alpha+alpha*(1./R)));
```

and, secondly, we need the derivative information

$$\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}} = \frac{dL_s(\mathfrak{X}_{ij})}{dx}\frac{d\mathfrak{X}_{ij}}{dr_{\mu\nu}} + \frac{\partial L_s(\mathfrak{X}_{ij})}{\partial r_{\mu\nu}}, \tag{3.8}$$

for every (s, i, j) and (μ, ν). Evidently, the first term in (3.8) is zero unless $(i, j) = (\mu, \nu)$ and the second term is zero unless $s = \nu$.

We have already prepared ourselves in Section 2 for the two kinds of derivatives of the 1-dimensional spline $L_s(x, ; r_{1s}, \ldots, r_{ms})$ involved in (3.8). The only difference here is that now we attain the derivative information in tensor form. The definition of `dL` via

```
dL = L;
dL.coefs(:,1) = zeros(size(L.coefs,1),1);
dL.coefs(:,2) = 3*L.coefs(:,1);
dL.coefs(:,3) = 2*L.coefs(:,2);
dL.coefs(:,4) = L.coefs(:,3);
```

for instance, characterizes simultaneously the ordinary derivatives of n 1-dimensional splines L_s with respect to the primal variable x for $s = 1, \ldots, n$. Thus for the case $(i, j) = (\mu, \nu)$, the first summation in (3.8) can be collectively calculated in compact form as follows:

```
tempdL = ppval(dL,frakX);
for s = 1:n
    FirstTerm(s,:,:) = squeeze(tempdL(s,:,:)) .* dfrakX.dR;
end
```

Note that, by construction, `FirstTerm` has the same tensor structure as $\mathfrak{R}$. See Figure 3.2.

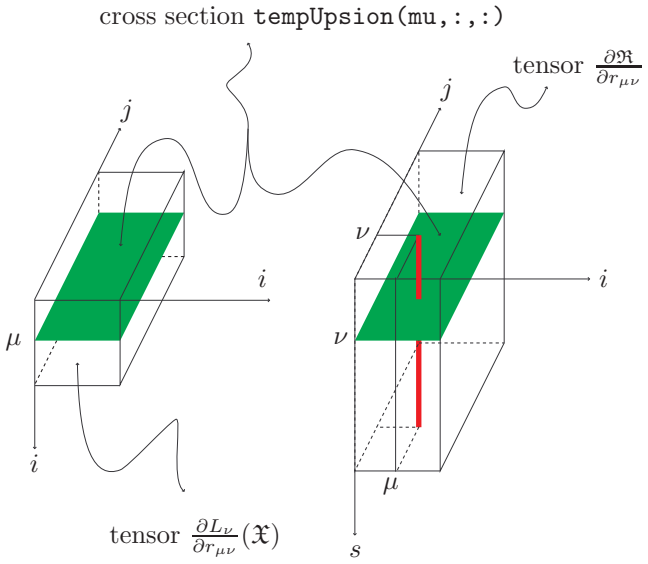

FIG. 3.3. *Diagram for the structure of* $\frac{\partial \mathfrak{R}}{\partial r_{\mu\nu}}$ *per fixed* (μ, ν).

On the other hand, we have argued in the preceding section that the sensitivity of L_ν to the change of the parameter $r_{\mu\nu}$ is precisely the spline that interpolates the data $\{(\mathbf{x}, \mathbf{e}_\mu)\}$, where $\mathbf{e}_\mu$ is the μth standard unit vector in $\mathbb{R}^m$. It is particularly useful to note that for $\mu = 1, \ldots, m$, the partial derivative $\frac{\partial L_\nu}{\partial r_{\mu\nu}}$ is the same for every $\nu = 1, \ldots, n$. Thus, taking advantage of the tensor operations, we issue the commands

```
Upsilon = spline(x,eye(m));
tempUpsilon = ppval(Upsilon,frakX);
```

to create an $m \times m \times n$ tensor whose μ-th horizontal slide, that is, the matrix `tempUpsilon(mu,:,:)` of size $m \times n$, denotes the evaluation $\frac{\partial L_\nu}{\partial r_{\mu\nu}}(\mathfrak{X})$ for every $\nu = 1, \ldots, n$. See the representation by the left graph in Figure 3.3. Note that there are as many horizontal layers, that is, sensitive matrices, as there are discrete grid points for capital.

For each fixed (μ, ν), we conclude from (3.8) that the tensor $\frac{\partial \mathfrak{R}}{\partial r_{\mu\nu}}$ is composed of the sum of a 2-dimension sensitive matrix `tempUpsilon(mu,:,:)` and a 1-dimensional array `FirstTerm(:,mu,nu)`. It is critical to note that the 2-dimensional slide `tempUpsilon(mu,:,:)`, and only this slide, is to be placed at the ν-th horizontal level in the tensor $\frac{\partial \mathfrak{R}}{\partial r_{\mu\nu}}$. Such a structure of $\frac{\partial \mathfrak{R}}{\partial r_{\mu\nu}}$ is depicted in the right graph of Figure 3.3, although there is no need to physically store this tensor.

It is interesting to envisage the motion that as (μ, ν) is changed, the vertical (red) bar in Figure 3.3 translates to a different location while the horizontal (green) cross section, copied from the μ-th plane of the left box, slides up or down in the s direction. This viewpoint turns out to be more advantageous for programming as we shall exploit below.

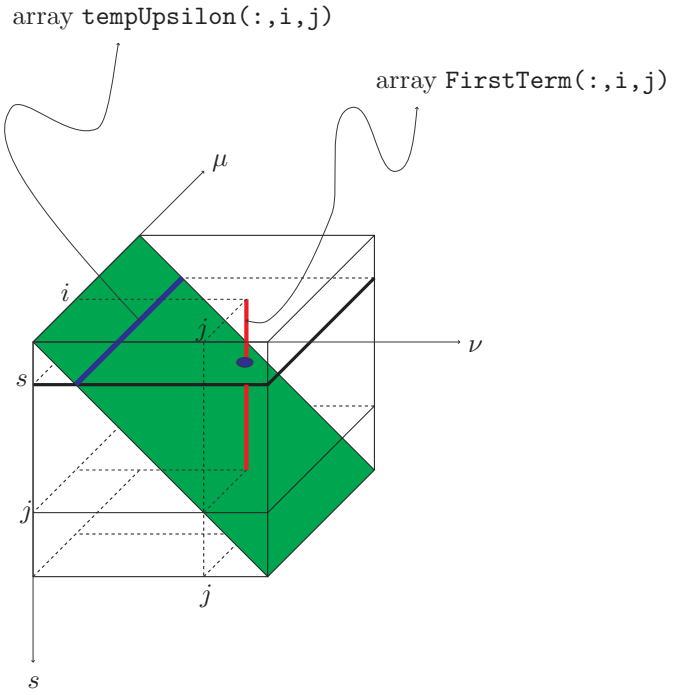

FIG. 3.4. *Diagram for the structure of* $\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}}$ *per fixed* (i,j).

The derivative $\frac{\partial F_{ij}}{\partial r_{\mu\nu}}$ is given by

$$\begin{aligned}\frac{\partial F_{ij}}{\partial r_{\mu\nu}} := -\beta \sum_{s=1}^{n} \pi_{js} \left(e^{y_s} \left(\frac{\mathfrak{X}_{ij}}{\mathfrak{R}_{sij}} \right)^{\alpha} \right)^{\zeta} (1-\mathfrak{R}_{sij})^{-\eta} \\ \left\{ e^{y_s} \left(\frac{\mathfrak{X}_{ij}}{\mathfrak{R}_{sij}} \right)^{\alpha-1} \left[(\alpha^2\zeta + \alpha(\alpha-1)) \left(\frac{\frac{d\mathfrak{X}_{ij}}{dr_{\mu\nu}}}{\mathfrak{X}_{ij}} - \frac{\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}}}{\mathfrak{R}_{sij}} \right) + \alpha\eta \frac{\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}}}{1-\mathfrak{R}_{sij}} \right] \right. \\ \left. + \alpha\zeta(1-\delta) \left(\frac{\frac{d\mathfrak{X}_{ij}}{dr_{\mu\nu}}}{\mathfrak{X}_{ij}} - \frac{\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}}}{\mathfrak{R}_{sij}} \right) + \eta(1-\delta) \frac{\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}}}{1-\mathfrak{R}_{sij}} \right\} \\ \underbrace{+ \left(e^{y_j} \left(\frac{x_i}{r_{ij}} \right)^{\alpha} \right)^{\zeta} (1-r_{ij})^{-\eta} \left(\frac{-\alpha\zeta}{r_{ij}} + \frac{\eta}{1-r_{ij}} \right)}_{\text{added only if } (i,j)=(\mu,\nu)}. \end{aligned} \tag{3.9}$$

To compute the "gradient" of F_{ij}, it is apparent that we need the quantity $\frac{\partial \mathfrak{R}_{sij}}{\partial r_{\mu\nu}}$ for fixed (s,i,j). Toward this, it is more useful to reinterpret Figure 3.3 as Figure 3.4. In other words, for each fixed (s,i,j), the $m \times n$ matrix $\frac{\partial \mathfrak{R}_{sij}}{\partial R}$ consists of only a 1-dimensional array `tempUpsilon(:,i,j)`, which is identical for all values of s, and a point `FirstTerm(s,i,j)`. Note that the $m \times n$ diagonal (green) plane in Figure 3.4 is made of identical columns `tempUpsilon(:,i,j)`.

We finally are ready to compute the gradient $\frac{\partial F_{ij}}{\partial R}$ for each fixed (i,j) and then the Jacobian matrix of F. We stress again that the partial derivative (3.9) can be assembled entry by entry, but to take advantage of the vector architecture in modern computers it is beneficial to first express $\frac{\partial F_{ij}}{\partial R}$ as

an $m \times n$ matrix in the following way. Then by reshaping each of these gradients into a $1 \times mn$ row vector, we obtain the $mn \times mn$ Jacobian matrix for $F(r_{11}, \ldots, r_{mn})$. We first define

```
capstone = (Q.^zeta) .* unoR.^(-eta) .* (eta./unoR - alpha*zeta*recipR);
```

which corresponds to the very last term in (3.9). For clarity of presentation and more so for economy of saving repeated computation, we group the expressions in (3.9) into four universal constants, two factors (as scaling vectors), two 1-dimensional arrays and two 2-dimensional arrays (to capture respectively the column vectors and the planes depicted in Figure 3.4 with different scaling). We then carry out the assembling process for the $mn \times mn$ Jacobian matrix `J` of F as follows.

```
c1 = alpha^2*zeta+alpha*(alpha-1);
c2 = alpha*eta;
c3 = alpha*zeta*(1-delta);
c4 = eta*(1-delta);

for j = 1:n
    for i = 1:m

        factor00 = exp(y).*(frakX(i,j)./frakR(:,i,j)).^alpha;
        factor01 = Pi(j,:)' .* factor00.^zeta .* (1-frakR(:,i,j)).^(-eta);
        factor02 = factor00.*(frakR(:,i,j)/frakX(i,j));

        vector01 = dfrakX.dR(i,j)/frakX(i,j) - FirstTerm(:,i,j)./frakR(:,i,j);
        vector02 = FirstTerm(:,i,j)./(1-frakR(:,i,j));
        plane01 = - tempUpsilon(:,i,j) * (1./frakR(:,i,j))';
        plane02 = tempUpsilon(:,i,j) * (1./(1-frakR(:,i,j)))';

        tempJ = (c1*plane01 + c2*plane02) * diag(factor02);
        tempJ = (tempJ +  (c3*plane01 + c4*plane02)) * diag(factor01);
        tempJ(i,j) = tempJ(i,j) + ((c1*vector01+c2*vector02).*factor02 + ...
            (c3*vector01+c4*vector02))'*factor01;

        tempJ = - beta * tempJ;

        tempJ(i,j) = tempJ(i,j) + capstone(i,j);

        J((j-1)*m+i,:) = reshape(tempJ, 1, m*n);

    end
end
```

Since the Jacobian matrix of F is now available, we may simply apply the Newton method to solve the (discrete) Euler equation.

4. Numerical Experiment. For numerical experiment, we adopt the parameters listed in Table 4.1. These parameters have been calibrated to reflect the key characteristics of the US economy. Their justifications can be found in [2] and the references therein.

It can be checked easily that, with the postulate that all technologies are essentially centered at

Parameter	β	η	θ	α	δ	ρ	σ
Value	0.9896	2.0	0.357	0.4	0.0196	0.95	0.0007

TABLE 4.1
Calibrated parameters

its zero mean, the general steady state solution of (3.2) is given by

$$\begin{aligned} k_{ss} &= \frac{\Psi}{\Omega + \phi\Psi}, \\ \ell_{ss} &= \phi k_{ss}, \\ c_{ss} &= \Omega k_{ss}, \end{aligned}$$

where the symbols stand for

$$\begin{aligned} \phi &:= \left(\frac{\frac{1}{\beta} - 1 + \delta}{\alpha} \right)^{\frac{1}{1-\alpha}}, \\ \Omega &:= \phi^{1-\alpha} - \delta, \\ \Psi &:= \frac{\theta(1-\alpha)\phi^{-\alpha}}{(1-\theta)}, \end{aligned}$$

Using our calibrated parameters, we have $k_{ss} \approx 23.1408$, $\ell_{ss} \approx 0.3105$, and $c_{ss} \approx 1.2883$. These steady state values provide guidance for selecting initial values and the boundaries of state variables. For example, in our numerical experiment we choose the interval $[0.7k_{ss}, 1.3k_{ss}]$ as the range of the capital. To approximate the autoregressive process by a finite dimensional Markov chain, we follow Tauchen's suggestion that the discretization is adequate for most purposes with $n = 9$.

Rate of convergence. We first demonstrate the rate of convergence of our method. A history of convergence for the Frobenius norm of $F(r_{11}, \ldots, r_{mn})$ is shown in Figure 4.1. Starting with the same constant initial value $R =$ `.4*ones(m,n)`, we report the convergence history for both the coarse grid $(m, n) = (4, 9)$ and the fine grid $(m, n) = (40, 9)$ in the left graph. When the initial value $R =$ `.7*ones(m,n)` is farther away from the steady state value of the labor supply ℓ_{ss}, we have observed with different grid sizes a behavior in the right graph. All cases seem to evidence the quadratic rate of convergence. The question is what limit points these iterations converge to.

Coarse grid. Concerning the limit points, we now demonstrate that the approximation of policy functions on coarse grid is as good as on fine grid by our spline approach. Plotted in Figure 4.2 are policy functions for (current) labor supply (3.1) and the next period capital (1.3) determined by our method. Specifically, all with constant initial value $r_{ij} = 0.4$, the graph at the left side of Figure 4.2 results from the application of our method to the coarse grid $(m, n) = (4, 9)$, whereas the result from the fine grid $(m, n) = (40, 9)$ is at on the right side. At the bottom of both graphs, the top drawings are plots (skeletons) of limit points returned by the Newton iteration and, hence, are of sizes 4×9 and 40×9, respectively. For clarity, we have lifted the drawings up by 20 units. The bottom drawings are plots after applying our 2-dimensional spline interpolation to a denser mesh with grid size at .01. The contour plots in the middle of Figure 4.2 manifest that the labor policy function is nonlinear. The colorbars on the right margins in all drawings help us to perceive the slops of the surfaces. The contour plots clearly indicate that the labor policy functions are somewhat different when using different grid sizes. This is worrisome! So we examine in further details the differences of the resulting policy functions between the coarse grid and the fine grid in Figure 4.3. In particular,

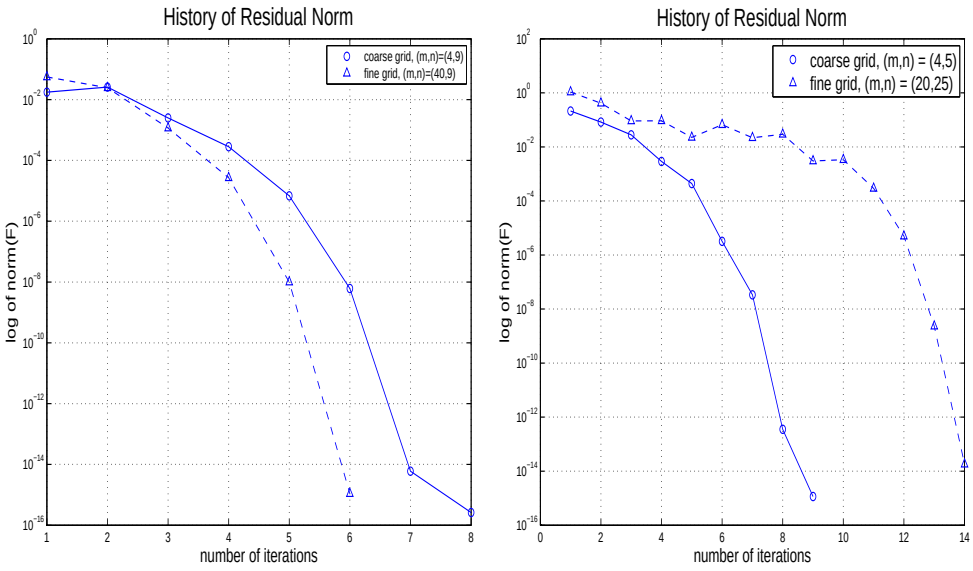

FIG. 4.1. *Convergence history of residuals with constant initial values* 0.4 *(left) and* 0.7 *(right) at all grid points.*

we see that the relative difference on the capital policy function differs at the order of 10^{-6}, strongly suggesting that using a coarse grid might be sufficient.

Multiple stationary points. We further our investigation of the limit points by comparing the difference between policy functions resulting from different initial values. On the left side in Figure 4.4 are the policy functions, or at least the limit points by our iteration, with initial values $R =$ `0.7*ones(12,15)`. The policy functions corresponding to $R =$ `0.4*ones(12,15)` are not shown because, as we have demonstrated earlier, they are similar to those in Figure 4.2. We immediate notice a significant difference between this new labor supply policy and that depicted in Figure 4.2, not just value-wise, but more so in the inclination of the slopes. A close examination of the contour plots of labor supply shows that the lower ends (blue color) are at opposite corners. The difference is caused more by the initial values than by the grid selections. On the right side of Figure 4.4, we plot the quantities of differences. It is interesting to note that, while the absolute difference between the policy functions of capital with different initial values is in the range $[-1.54, -1.04]$, the relative difference between these two policy functions of capital is around 5%. Would such a dissimilarity make a significant impact on the interpretation of economic dynamics? Mathematically, at least, this finding manifests an important fact — the existence of other critical points satisfying the Euler equation, namely, the first order optimization condition. We need to investigate the role of these "other" critical points. In particular, we want to show that they are not optimal policy functions[3].

In classical constrained optimization, a standard approach to tell the character of a stationary point is to check the second order optimality condition, that is, the definiteness of the projected Hessian of the objective function at this critical point. For dynamic programming, such a realization is hard to come by. Instead, we check the value function corresponding to each proposed policy function.

Value function iteration. If the resulting limit point $p(k_t, z_t)$ of the Newton iteration is a true

[3]It is a known fact that, for neoclassical growth models, the policy function for capital is monotone non-decreasing. We do observe in Figure 4.4 that, while the policy function for labor supply is non-increasing, the corresponding policy function for capital is indeed monotone non-decreasing. So the visualization of monotonicity of these critical points does not help to rule out that they are wrong policy functions.

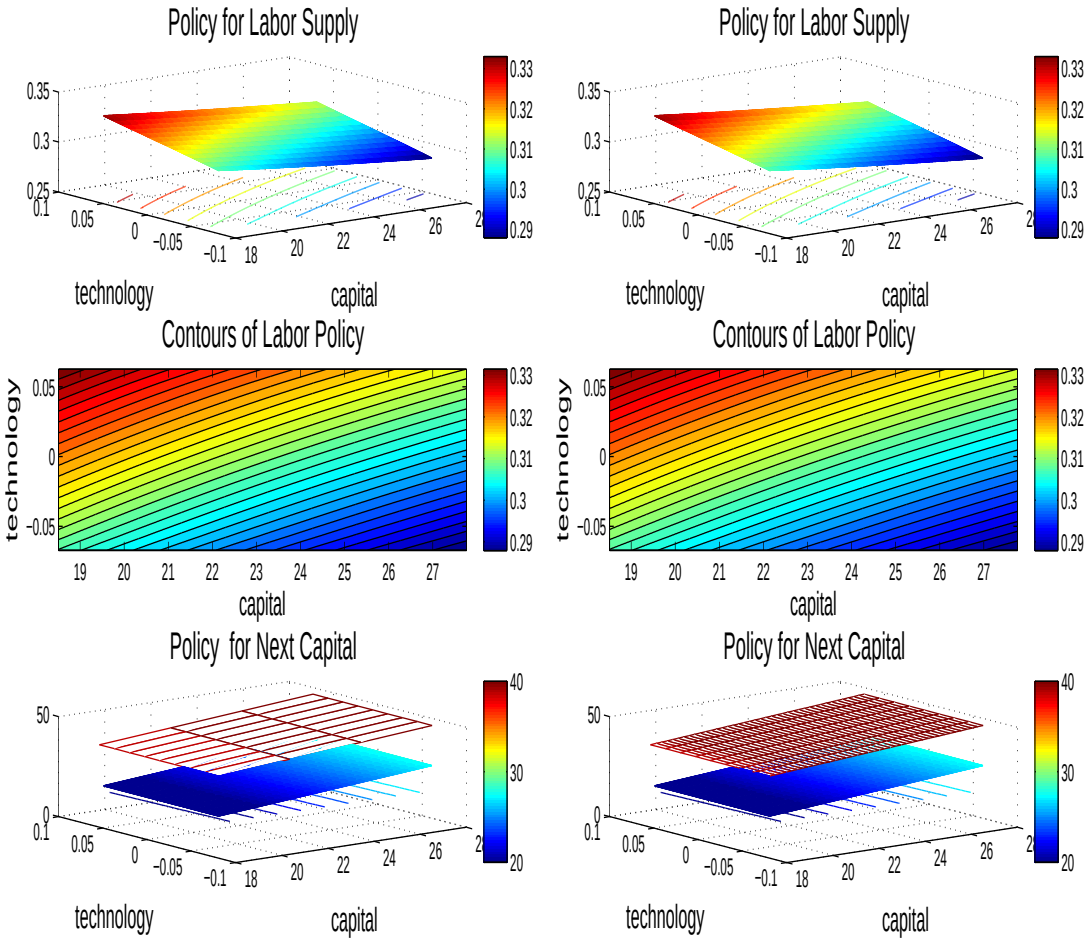

FIG. 4.2. *Comparison of spline surfaces. Left: coarse grid* $(m, n) = (4, 9)$*; Right: fine grid* $(m, n) = (40, 9)$*; constant initial value* $r_{ij} \equiv 0.4$.

policy function, then by its definition there should exist a corresponding value function satisfying the Bellman equation

$$v(k_t, z_t) = u(c_t) + \beta \mathcal{E}_t \left[v(p(k_t, z_t); z_{t+1}) \right], \tag{4.1}$$

where the maximization required in the original (1.2) is automatically fulfilled. We may employ the conventional scheme of value function iteration, without the burden of computing the transitional policy functions (maximizers), to find the value function. This is a good news because selecting the maximizer usually is the most time-consuming step in the value function iteration and hereby we already have a fast method that computes the policy function to high precision. We also are less concerned about acceleration [2, 5] because, as our experiments have persistently demonstrated, coarse grids followed by spline interpolation are general sufficient.

Assuming that the $m \times n$ matrix `R` is a limit point returned by the Newton method applied to the Euler equation, a prototype of the value function iteration is exemplified below. The neatness of using tensors to represent multiple layers of surfaces and their interactions should be manifestly clear from the compactness of the code.

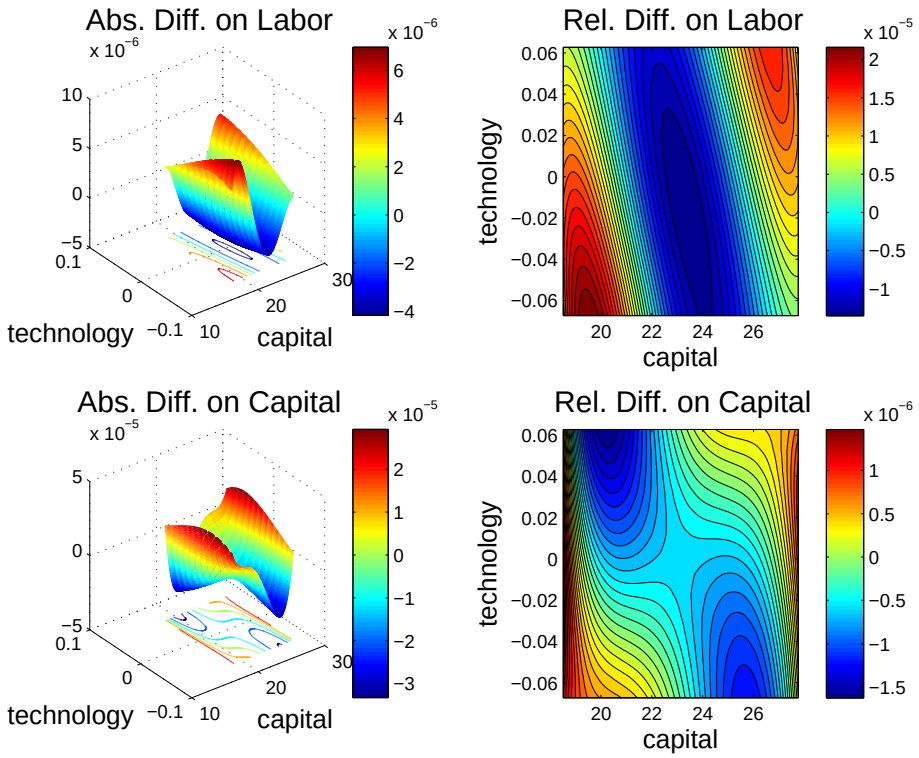

FIG. 4.3. *Differences between policy functions with same initial value* $r_{ij} \equiv 0.4$ *but different grid sizes* $(m, n) = (4, 9)$ *vs.* $(40, 9)$.

```
frakC = theta*(1-alpha)/(1-theta)*Q.*unoR;
ufrakC = (((frakC.^theta) .* unoR.^(1-theta)).^(1-eta))/(1-eta);

vold = -1000*ones(m,n);
vnew = ((c_ss^theta)*(1-ell_ss)^(1-theta))^(1-eta)/(1-eta)*ones(m,n);

for iter = 1:10000
    if (norm(vold-vnew) <= norm(vnew)*m*n*eps)
        display('Value function obtained. Returned!')
        break
    else
        vold = vnew;
        V = spline(x,vnew');
        tempV = ppval(V,frakX);
        vnew = zeros(m,n);
        for s = 1:n
            vnew = vnew + squeeze(tempV(s,:,:)) * diag(Pi(:,s));
        end
        vnew =  ufrakC + beta * vnew;
    end
end
display('Warning --- No break occurs yet for convergence')
```

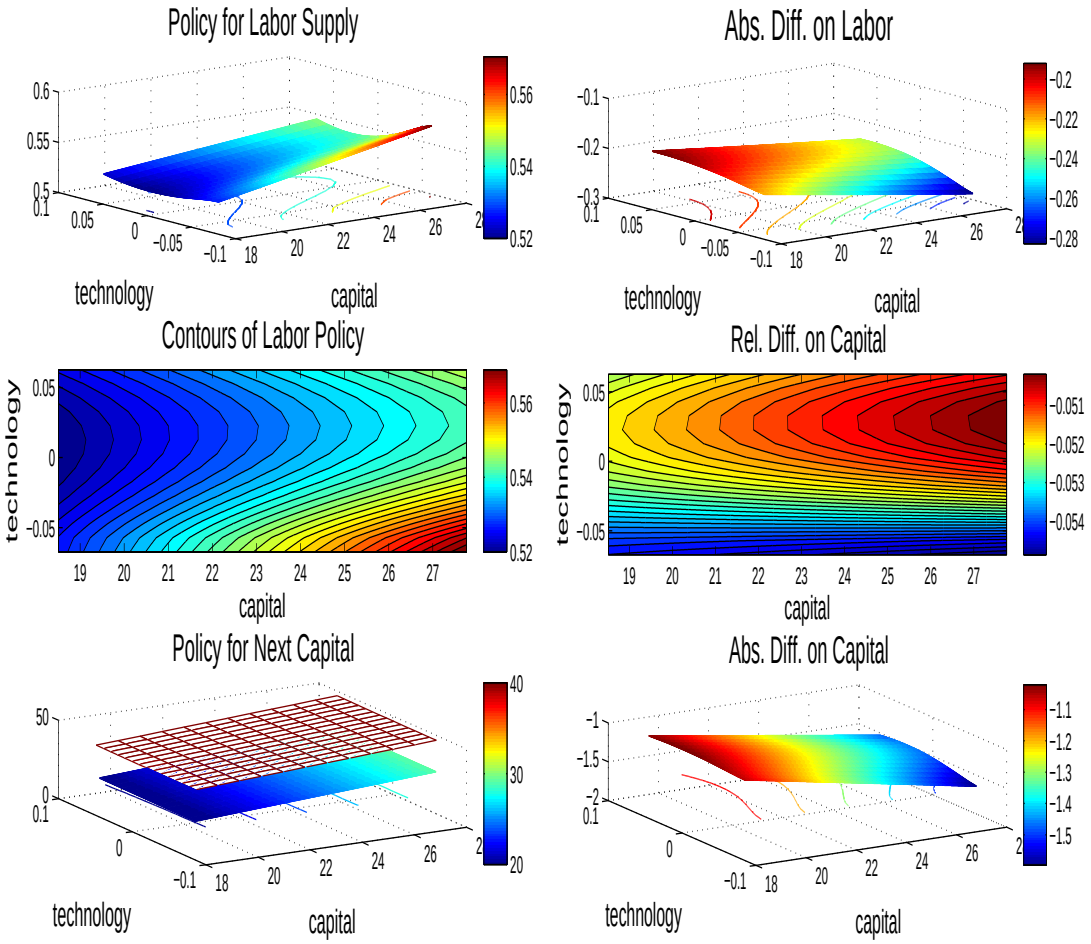

FIG. 4.4. *Left: policy functions with initial values* $r_{ij} \equiv 0.7$ *on grid* $(m, n) = (12, 15)$*; Right: differences between policy functions with different initial values* $r_{ij} \equiv 0.4$ *vs.* 0.7.

We remark that the maximal allowable number `iter` of iterations above is chosen circumspectly because of the concern that the nearness of our $\beta = 0.9896$ to one is expected to cause slow contraction. We caution also that the prototype scheme above has not taken into account any acceleration tactics at all. The scheme here is only to demonstrate how the value function could be found once we have the policy function in hand.

What we have obtained is the value function depicted in Figure 4.5 for the initial value $R =$ `.4*ones(4,9)` followed by our 2-dimensional spline interpolation, which is as good as using finer grids. On the other hand, we have observed from our experiments that the "other policy function" found by using, say, the initial value $R =$ `.7*ones(12,15)`, does *not* lead to convergence at all by the value function iteration. Indeed, the value function blows up to infinity in just a few iterations. The failure is because these other critical points lack the property of maximization needed in (1.2). A 5% relative discrepancy of the capital policy at every time step leads to a catastrophe!

Euler equation error function. We have already reported that our method converges quadratically and returns near machine-precision residual values of F, but the nonlinear system (3.6) is only a

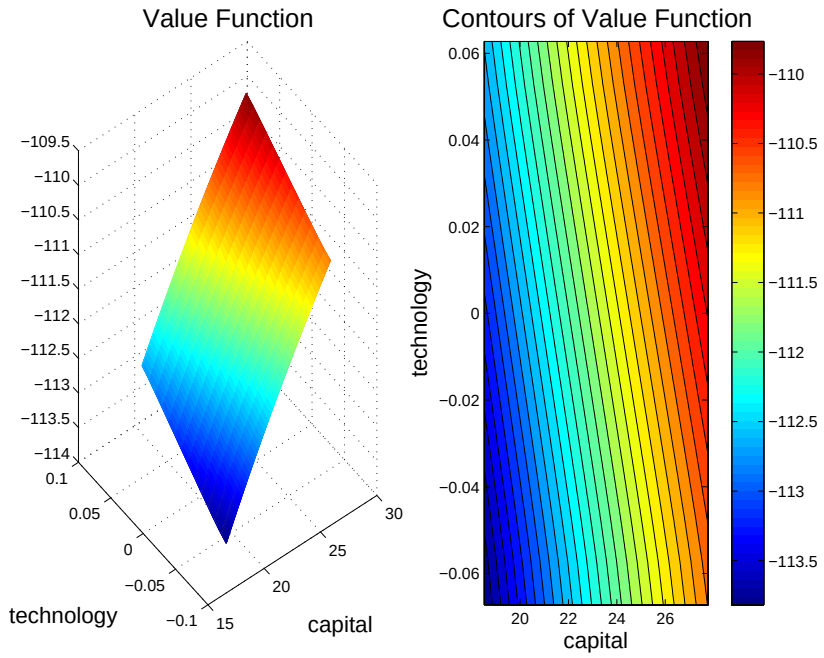

FIG. 4.5. *Value function based on the grid size* $(m, n) = (4, 9)$ *with initial value* 0.4.

discrete approximation to the original Euler equation. Though by the continuity argument we expect that precision would prevail at off-grid points, we still prefer to assure unit free accuracy in the Euler equation throughout the domain of interest quantitatively. A commonly used assessment tool is to compute the so called Euler equation error function defined by [10]

$$E(k_t, z_t) := 1 - \frac{\left\{ \frac{\beta \mathcal{E}_t \left[\left(c_{t+1}^{\theta} (1-\ell_{t+1})^{1-\theta} \right)^{1-\eta} / c_{t+1} \left(\alpha e^{z_{t+1}} k_{t+1}^{\alpha-1} \ell_{t+1}^{1-\alpha} + 1 - \delta \right) \right]}{(1-\ell_t)^{(1-\theta)(1-\eta)}} \right\}^{1/\zeta}}{c_t}, \tag{4.2}$$

which has the meaning of "the relative optimization error incurred by the use of the approximated policy rule". The absolute values of the Euler equation errors at the limit points by our method on two grid sizes are plotted in Figure 4.6 on the logarithmic (base 10) scale. As expected, the downward spikes occur at the grid points of capital. The precision of our result, even when using coarse grids, should be quite obvious.

We have pointed out earlier that the limit point corresponding to this particular initial value $R =$ `0.7*ones(12,15)` fails to produce a proper value function. It might be curious to ask whether such a failure could have been detected in the Euler equation errors. We calculate the corresponding error function in Figure 4.7. Note that the Euler equation errors persistently maintain reasonable precision throughout the domain of interest, strongly suggesting that merely checking the Euler equation errors is not enough to qualify a policy function.

Robustness. Finally, we repeat the robustness tests conducted in [2] by keeping all parameters unchanged except that we increase the risk aversion from $\tau = 2$ to $\tau = 50$ and the standard deviation of the technology shock from $\sigma = 0.007$ to $\sigma = 0.035$. Under such a situation of extreme calibration, a nature consequence is the accumulation of capital. Thus we have to modify the capital interval, after several trials and errors, from $[0.7k_{ss}, 1.3k_{ss}]$ to, say, $[20, 60]$. Test results are reported in Figure 4.8. It is seen that the quadratic convergence occurs only at a much later stage of iterations, but once

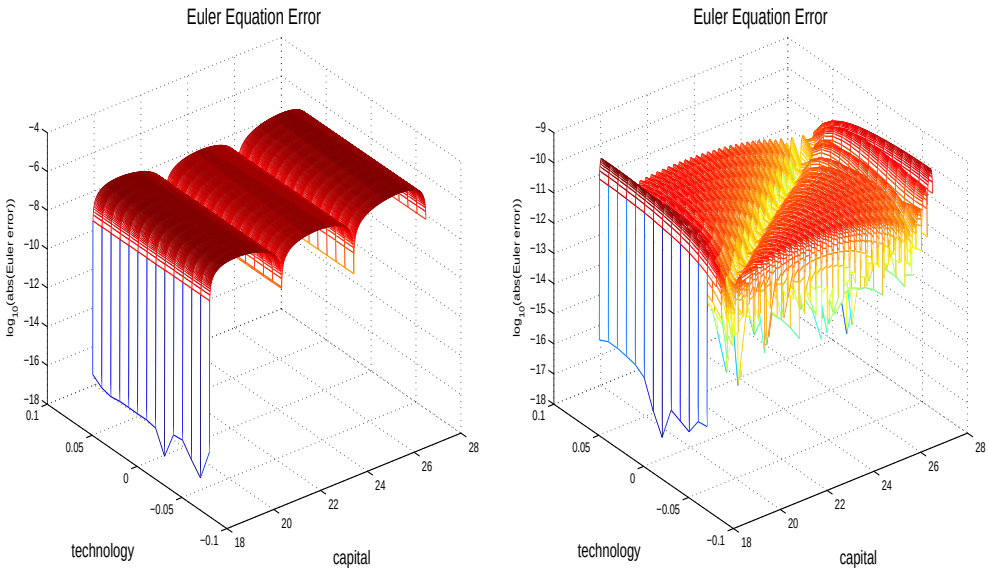

FIG. 4.6. *Euler equation errors based on the grid sizes* $(m, n) = (4, 9)$ *and* $(m, n) = (40, 9)$ *with initial value* 0.4.

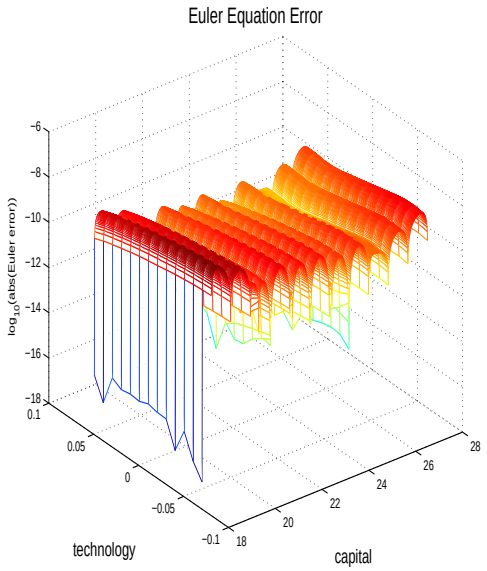

FIG. 4.7. *Euler equation errors based on the grid sizes* $(m, n) = (12, 15)$ *with initial value* 0.7.

convergence occurs we achieve the machine precision. Even though we are using a relative coarse grid $(m, n) = (20, 9)$, the overall Euler errors are superior than any of those methods reported in [2]. One peculiar phenomenon is that the initial value $R =$ `0.4*ones(m,n)` that works well for the previous benchmark problem does *not* lead to convergence at all for this extreme calibrated problem. Test runs complain that the Jacobian matrix is nearly singular. The results reported in Figures 4.8 are obtained with $R =$ `0.8*ones(20,9)`. Of course, having learned from previous studies, we are

concerned whether this limit point is a false policy function. We carry out the value function iteration described earlier and obtain convergence as is shown in Figure 4.9. Our numerical experiments with other intermediate cases suggest that the odd-looking shape of the value function is more the effect of high risk aversion than high variance.

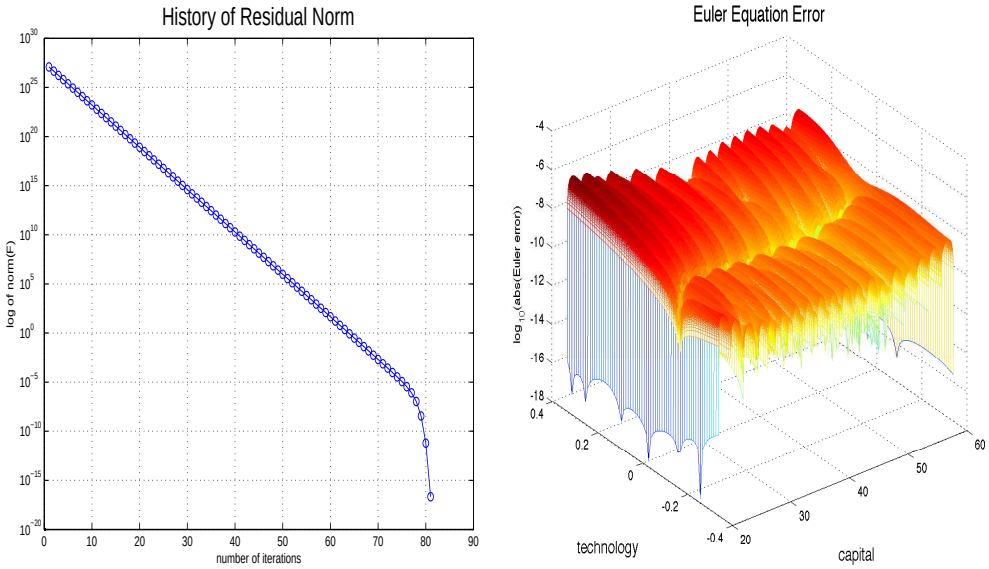

FIG. 4.8. *History of residual norm and Euler equation errors based on the initial value* $0.8 * ones(20, 9)$ *for the case* $\tau = 50$ *and* $\sigma = 0.035$.

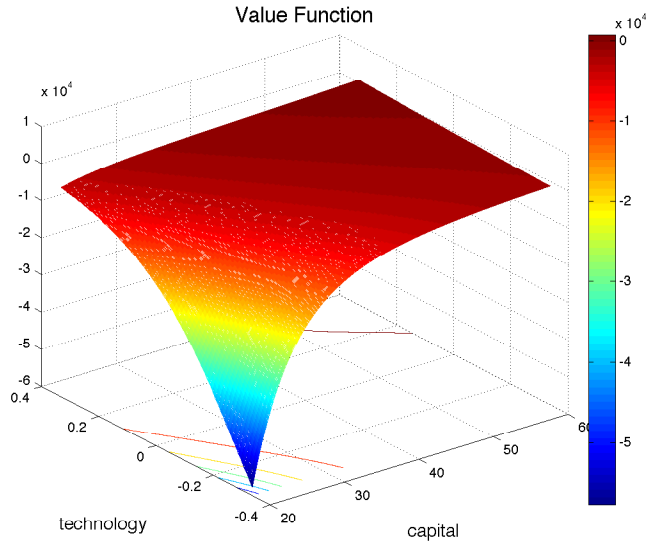

FIG. 4.9. *Value function based on the initial value* $0.8 * ones(20, 9)$ *for the case* $\tau = 50$ *and* $\sigma = 0.035$.

5. Conclusion. We propose utilizing composite cubic splines to approximate policy functions in economic dynamics. We use the neoclassical growth model with leisure choice to demonstrate the working of our ideas. The major advantage is that this 2-dimensional spline is essentially a one 1-dimensional spline weighted by another 1-dimensional spline. Hence we benefit directly from the ease of derivative calculation and tensor manipulation of 1-dimensional splines, which allows us to perform extensive but fast simulations. Of particular importance is that, in contrast to [2, Remark 11, p. 2490], using coarse grids seems sufficient to approximate policy functions with high degree of precision. Also, since we are dealing with the policy functions, not the value functions, shape-preserving [12, 14] seems immaterial.

We are able to locate, fast and up to the machine precision, multiple critical points for the (discrete) Euler equation. This discovery strongly suggests that, without a second order optimality check or its like, merely having the convergence by conventional projection methods might suffer from misleading to wrong policy functions. Fast computation with high precision of convergence thus is of practical relevance because it allows us to effectively repeat simulations for different parameter values in search of the correct equilibrium path for the economy [2]. Our method seems capable of doing that.

Finally, as our composite splines connect grid points into smooth surfaces, the finite Markov chain approximation to the autoregressive process could be replaced by quadratures. More importantly, the tensor operations we have carried out in this presentation is based on the composite rule which ultimate reduces the calculation to the advantage of the 1-dimensional cubic spline. This composite rule should be in principle generalizable to multi-agent models. Whether we continue to have the asset of using coarse grids in the higher dimensional problems remains to be further investigated.

REFERENCES

[1] J. Adda and R. W. Cooper, *Dynamic economics: quantitative methods and applications*, The MIT Press, Cambridge, MA, 2003.

[2] S. B. Aruoba, J. Fernández-Villaverde, and J. F. Rubio-Ramírez, *Comparing solution methods for dynamic equlibrium economies*, J. Econom. Dynam. Control, 30 (2006), pp. 2477–2508.

[3] R. E. Bellman, *Dynamic programming*, Princeton Landmarks in Mathematics, Princeton University Press, Princeton, NJ, 2010. Reprint of the 1957 edition, With a new introduction by Stuart Dreyfus.

[4] D. P. Bertsekas, *Dynamic programming and stochastic control*, Academic Press, New York, 1976.

[5] C.-S. Chow and J. N. Tsitsiklis, *An optimal one-way multigrid algorithm for discrete-time stochastic control*, IEEE Trans. Automat. Control, 36 (1991), pp. 897–914.

[6] L. Cooper and M. W. Cooper, *Introduction to dynamic programming*, vol. 1 of International Series in Modern Applied Mathematics and Computer Science, Pergamon Press, Oxford, 1981.

[7] J. W. Daniel, *Splines and efficiency in dynamic programming*, J. Math. Anal. Appl., 54 (1976), pp. 402–407.

[8] C. de Boor, *A practical guide to splines*, vol. 27 of Applied Mathematical Sciences, Springer-Verlag, New York, revised ed., 2001.

[9] B. Heer and A. Maussner, *Dynamic general equilibrium modelling*, Springer, Berlin, 2005. Computational methods and applications.

[10] K. L. Judd, *Projection methods for solving aggregate growth models*, J. Econom. Theory, 58 (1992), pp. 410–452.

[11] ———, *Numerical methods in economics*, MIT Press, Cambridge, MA, 1998.

[12] K. L. Judd and A. Solnick, *Numerical dynamic programming with spahe-preserving splines*, tech. report, Hoover Institute, Stanford University, 1994. preliminary and incomplete, `http://bucky.stanford.edu/papers/dpshape.pdf`.

[13] E. Laporte and P. Le Tallec, *Numerical methods in sensitivity analysis and shape optimization*, Modeling and Simulation in Science, Engineering and Technology, Birkhäuser Boston Inc., Boston, MA, 2003.

[14] L. L. Schumaker, *On shape preserving quadratic spline interpolation*, SIAM J. Numer. Anal., 20 (1983), pp. 854–864.

[15] ———, *Computing bivariate splines in scattered data fitting and the finite-element method*, Numer. Algorithms, 48 (2008), pp. 237–260.

[16] N. L. Stokey and R. E. Lucas, Jr., *Recursive methods in economic dynamics*, Harvard University Press, Cambridge, MA, 1989. With the collaboration of Edward C. Prescott.

[17] G. Tauchen, *Finite state markov-chain approximations to univariate and vector autoregressions*, Econom. Lett., 20 (1986), pp. 177–181.