Value Function Approximation for Dynamic Game Models

Wyatt Jones

University of Iowa

August 30, 2017

1 Introduction

The analysis of dynamic industrial organization (I.O.) models is limited by the computational burden of computing equilibrium policies. These policies are typically computed by solving for the fixed point of a function whose domain is a subset of $\mathbb{R}^n$, where n is the dimension of the state space (Pakes & McGuire, 1992). In dynamic I.O. models, the dimension of the state space is typically the number of potentially active firms. In the Ericson and Pakes (1995) framework, each firm's state is an element of a discrete grid with m possible values. If we impose that the policy functions are symmetric, then the number of elements in the state space grows exponentially in the number of states per firm. Additionally, if there are multiple decision variables, the number of objective function evaluations required to find the optimal policy grows exponentially in the number of decision variables. These curses of dimensionality in the state and action spaces limit the applications that the Ericson and Pakes (1995) framework can be applied to (Aguirregabiria and Nevo (2013); Doraszelski and Judd (2007); Pakes and McGuire (2001); Weintraub, Benkard, and Van Roy (2008)).

I show adaptations to the existing methodology that may circumvent these computational problems for Markov Perfect models in which the size of the state space or action space is large. This is because my adaptations are able to overcome the issue with approximating a

highly non-linear or even discontinuous value function and thus allow the algorithm to use a small subset of the state space to approximate the rest of the value function.

In this paper, I show methods for reducing the computational cost imposed by the curse of dimensionality in the action and state space that occurs when solving for a Markov Perfect Equilibrium (MPE). The rest of the paper is organized as follows. In Section 2, I outline the dynamic industry model and describe the relationship between my work and previous literature. In Section 3, I show methods for avoiding the curse of dimensionality in the action space. In Section 4, I show methods for avoiding the curse of dimensionality in the state space, the difference between using multidimensional Chebyshev polynomials and using Artificial Neural Nets (ANN) for approximation, and issues that arise when trying to use the gradient of the value function for approximations. Finally, Section 5, presents conclusions and directions for future research.

2 Model

In order for their model to be tractable, Ericson and Pakes (1995) make assumptions about how firms interact. In an industry with n firms, each firm is assumed to have a single dimensional state variable ω_j, where $j \in \{1, \ldots, n\}$, that determines the relative position of a firm in an industry compared to the strength of competition outside of the industry. States are assumed to be discrete and typically $\omega_j \in \mathbb{Z}_+$. This assumption reduces the multidimensional heterogeneity that exists between firms in the real world to a single dimension. Additionally, the discretization of the state space reduces the accuracy with which industry dynamics can be represented.

Let $S = \{\omega_1, \ldots, \omega_n\} \in \mathbb{Z}_+^n$ denote the industry state. In equilibrium it is assumed that current profits for firm j are weakly decreasing in S and weakly increasing in ω_j. Thus increasing my own state improves my position in the market and if other firms increase their own state, my current period profits will decrease.

The state S changes stochastically conditional on the actions of every firm in the industry and firms' choose their actions to maximize the expected present discounted value of profits as a function of the state S. In order to further reduce the complexity of the framework

Ericson and Pakes (1995) make the necessary assumptions to assure that the state follows a Markov process. Thus firms are restricted to Markovian strategies and do not directly observe the actions of the other firms.

While these assumptions limit the frameworks ability to generate the variation observed in the data they are necessary to construct a more tractable model however, even after these assumptions it is not computationally tractable to model industries with more than a few firms (Weintraub et al., 2008). In order to further reduce the complexity of the framework so that it is computationally tractable, Doraszelski and Satterthwaite (2010) additionally impose symmetry restrictions. By adding additional structure to the framework they prove the existence of a symmetric equilibrium in pure strategies.

Firm j's value function when in industry state S will be expressed by

$$V_j(S) = \max_{\mu_j \in \Gamma(S)} \pi(S, \mu_j, \mu_{-j}) + \beta \mathbb{E}[V(S')|S, \mu_j, \mu_{-j})]$$

where μ_j is the policy of firm j which will could include the pricing, investment, and entry or exit decision, $\Gamma(S)$ is the set of feasible actions in state S, μ_{-j} is the policies of all other firms other than firm j, and S' is the state transitioned to conditional on S, μ_j and μ_{-j}.

Throughout this paper I will refer to a numerical example that is the same as the example used in Doraszelski and Judd (2012). The example is a simplified version of the Pakes McGuire quality ladder model (Doraszelski & Judd, 2012). I will focus on the case where there are two firms choosing price and investment. The state variable $S = \{\omega_1, \omega_2\}$ represents the vertically differentiated quality of their product. Let $\omega_j \in \{1, \ldots, 18\}$ and the approximation nodes $S_i = \{\omega_{1,i}, \omega_{2,i}\} = \{\lfloor \frac{i}{18} \rfloor, i \mod 18\}$ for $i = 1, \ldots, 324$. Consumer l's utility function for product j is given by $u_{lj} = g(\omega_j) - p_j + \epsilon_{lj}$ where ϵ_{lj} is an extreme type 1 error. The function $g(\omega)$ maps the discrete states, ω_j, to the quality consumers perceive for product j. Both Pakes and McGuire (1992) and Doraszelski and Judd (2012) used the function

$$g(\omega) = \begin{cases} 3\omega - 4 & \text{if } \omega \leq 5 \\ 12 + \ln(2 - \exp(16 - 3w)) & \text{if } \omega > 5 \end{cases}$$

however as Figure 1 shows this function is not continuous between 5 and 6. This causes a

problem when attempting to approximate the model with a continuous state space. Therefore I will use the function

$$g(\omega) = \frac{14.7}{1 + \exp(4 - 1.2\omega)}$$

which Figure 1 shows is continuous and a close approximation of the original function.

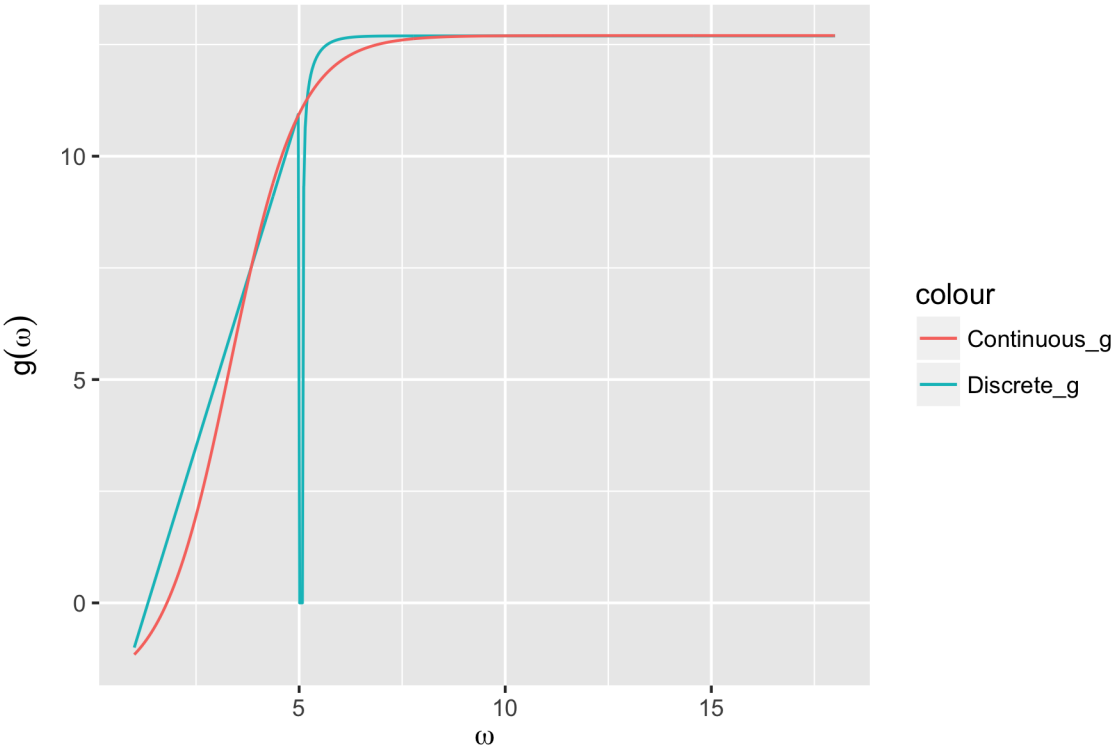

Figure 1: Plot of both versions of $g(\omega)$

The single period objective function for firm 1 as a function of the industry state, S, the actions of firm 1, μ_1, and the other firms' actions, μ_2, can be expressed with the following equation. Let $\mu_j = \{p_j, x_j\}$, where p_j represents the price charged to consumers for product j and x_j represents the level that firm j invests into the quality of their product. M represents the market size, and c represents the marginal cost of production. The single period payoff function is

$$\pi_1(S, \mu_1, \mu_2) = M \frac{\exp(g(S_1) - p_1)}{1 + \exp(g(S_1) - p_1) + \exp(g(S_2) - p_2)}(p_1 - c) - x_1$$

In order to calculate the expected future value for a policy μ_1 let

$$\mathbb{E}[\hat{V}(S'|S,\mu_1,\mu_2)] = \sum_{\omega_1'=1}^{m}\sum_{\omega_2'=1}^{m}\hat{V}(\{\omega_1',\omega_2'\})\Pr(\omega_1'|\omega_1,x_1)\Pr(\omega_2'|\omega_2,x_2)$$

Where the probability of transitioning to the state ω' given investment x and current state ω is given by

$$\Pr(\omega'|\omega,x) = \begin{cases} \frac{(1-\delta)\alpha x}{1+\alpha x} & \text{if } \omega' = \omega+1 \\ \frac{1-\delta+\delta\alpha x}{1+\alpha x} & \text{if } \omega' = \omega \\ \frac{\delta}{1+\alpha x} & \text{if } \omega' = \omega-1 \end{cases}$$

$$\Pr(\omega'|1,x) = \begin{cases} \frac{(1-\delta)\alpha x}{1+\alpha x} & \text{if } \omega' = 2 \\ \frac{1+\delta\alpha x}{1+\alpha x} & \text{if } \omega' = 1 \end{cases}$$

$$\Pr(\omega'|L,x) = \begin{cases} \frac{1-\delta+\alpha x}{1+\alpha x} & \text{if } \omega' = L \\ \frac{\delta}{1+\alpha x} & \text{if } \omega' = L-1 \end{cases}$$

where δ represents the probability that the outside alternative increases in quality, and α represents the efficacy of investment in generating increases in efficiency. Through out the paper I will reference the MPE of this model found when $\beta = 0.925$, $M = 5$, $c = 5$, $\alpha = 3$, and $\delta = 0.7$.

2.1 Pakes-McGuire algorithm

The Pakes and McGuire (1992) algorithm is an iterative backward solution method which uses numbers stored in memory from the previous iteration to calculate an update to those numbers and runs until the updated numbers satisfy a convergence condition. At the start of the algorithm N approximation nodes, $\{S_i\}_{i=1}^{N}$, must be chosen. Each node represents a possible industry state that can be visited in the model. It is also necessary to initialize an $\epsilon > 0$ for the convergence criterion and to initialize the value function.

The algorithm uses the assumption that firms use symmetric strategies so that only one value function needs to be calculated. For notational purposes I will denote the firm for which

the value function is calculated as firm 1. Since we are interested in finding only symmetric MPE, the Pakes and McGuire (1992) algorithm uses the calculation of firm 1's policies in iteration k-1 to determine the policies of all other firms in iteration k. For example, in the numerical example the pricing policy for firm 2 in state S_i, and iteration k is represented as $p^k_{i,2} = p^{k-1}_{i',1}$ where i' is such that if $S_i = \{\omega_1, \omega_2\}$ then $S_{i'} = \{\omega_2, \omega_1\}$.

More generally, for each iteration k the algorithm uses firm 1's policy from last iteration, μ_1^{k-1} to determine the policy that other firms will choose, μ^k_{-1}, and then calculates μ_1^k based on μ^k_{-1} and v_i^{k-1}. Then the algorithm checks a convergence condition by evaluating the distance between the previous iterations value function and the current value function.

Algorithm 1 Pakes McGuire Algo

procedure Choose N approximation nodes $\{S_i\}_{i=1}^N$, error tolerance for value function updating ϵ.
  Let $v_i^0 = 0$ for $i = 1, \ldots, N$
  while $\left\|v_1^k - v_1^{k-1}\right\| > \frac{\epsilon(1-\beta)}{2\beta}$ **do**
    for $i = 1, \ldots, N$ **do**
      Solve the Bellmann equation and store in memory $v^k_{i,1}$ and $\mu^k_{i,1}$

$$v^k_{i,1} = \max_{\mu^k_{i,1} \in \Gamma(S_i)} \pi(S_i, \mu^k_{i,1}, \mu^{k-1}_{i,-1}) + \beta \mathbb{E}[\hat{V}(S') | S_i, \mu^k_{i,1}, \mu^{k-1}_{i,-1})]$$

      Let $k = k + 1$
    end for
  end while
end procedure

The run time of this algorithm can be approximated with the following formula. Let K represent the number of value function iterations required for convergence, N the number of states in the state space, and f_{evals} be the number of function evaluations required to find the optimal policy.

$$\text{run time} = (K)(N)(f_{evals})(\text{Computation time of f})$$

$$H(S_i) = \max_{\mu^k_{i,1} \in \Gamma(S_i)} f(S_i, \mu^k_{i,1}) \tag{1}$$

$$f(S_i, \mu^k_{i,1}) = \pi(S_i, \mu^k_{i,1}, \mu^{k-1}_{i,-1}) + \beta \mathbb{E}[\hat{V}(S')|S_i, \mu^k_{i,1}, \mu^{k-1}_{i,-1})]$$

This algorithm suffers from several computational issues. There are curses of dimensionality in both the state space and action space that can cause the algorithm to take too long to run. For a fixed number of iterations required for convergence, which I will denote as K, if the industry of interest has just 10 firms then N can be too large. In the numerical example in this paper, each firm has 18 possible states and so for 10 firms, the size of the state space would be $18^{10} = 3,570,467,226,624 = N$. If the time it takes to compute $f(S, \mu)$ takes just 0.01 seconds then it would take 413,248 days to just evaluate the function once in each state. There are many different ways to decrease the computational cost imposed by increasing the number of firms, and in Section 4 I will show several methods of value function approximation to overcome this issue.

In applications of the Ericson and Pakes (1995) framework firms can have many decision variables. Typically a firm chooses the price of their product, how much to invest to improve their state, and whether to enter or exit, however many firms can have a much more complex decision making process. In the numerical example used in this paper firms just choose the price of their product and level of investment to improve the quality of their product. In order to choose the optimal action using a simple grid search procedure with a 1e-5 degree of accuracy there would need to be a 400,000 by 500,000 grid representing each possible choice that the firm could make (since in equilibrium the optimal price varies from 6 to 11 and the optimal level of investment varies from 0 to 4). Thus, the number of function evaluations per state per iteration, which I will denote as f_{evals}, would be 200,000,000,000. An algorithm implemented this way would take 23,148 days to find the optimal action in a single state in a single iteration. There are many different methods that allow the optimal action to be found with a high degree of accuracy very quickly and Section 3 discusses how to use these methods to decrease f_{evals}.

In many applications the run time required to compute an equilibrium determines the

way in which the theoretical model was written (Benkard, Jeziorski, & Weintraub, 2015; Doraszelski & Pakes, 2007). Often, this leads to including only 2-4 firms, or using a profit function that can be computed outside the algorithm's main loop to reduce the number of decision variables.

Thus in order to develop richer models of industry dynamics that are a closer representation of reality, it is important to understand the factors that determine a models run time and the methods that can be used to make a model computationally tractable.

3 Avoiding the curse of dimensionality in action space

In order to reduce the computational cost caused by a curse of dimensionality in the action space it is necessary to either reduce the number of decision variables, or reduce the number of function evaluations required to find the policy that maximizes the objective. In the Ericson and Pakes (1995) framework it is common to use a static-dynamic breakdown to reduce the number of decision variables. A static-dynamic breakdown means that the model is constructed in such a way that in the dynamic industry it is optimal for firms to choose the same prices that they would choose in a single period model. If this is the case then the profitability for firm 1 in each state only has to be computed once and then stored in memory. In some settings it is not desirable to use a static-dynamic breakdown since the pricing decision does influence the dynamics of the industry. This is the case in dynamic network industries where the price of a product in a period t affects the market share for each firm in t+1 (Chen & Doraszelski, 2006; Chen, Doraszelski, & Harrington Jr, 2009; Mitchell & Skrzypacz, 2006).

If there several decision variables, a high degree of accuracy is required, or the run time is too large it is advantageous to use more sophisticated methods to find the optimal policy in a state. The first issue that arises is if the decision variable is continuous. Exit and entry decisions are naturally modeled as a discrete 0 for exit and 1 for remaining in the industry, but many other policy decisions, such as how much to invest, or the price charged for a product are most naturally modeled as a continuous variable. Since in the Ericson and Pakes (1995) framework the state space is discretized in order for $H(S)$ to be well defined it

is necessary to map continuous actions to discrete future states. This is because the objective function $f(S, \mu)$ contains the evaluation of the expected value of future payoffs conditional on the current state and actions. In this expectation the value function $V : S' \in \{S_i^N\} \to \mathbb{R}$ can only be evaluated if $S' \in \{S_i^N\}$ where $S' = h(S, \mu)$. The transition function $h(S, \mu)$ maps the policy μ and discrete states S to discrete states S'. In the numerical example used in this paper the transition function represents the change in quality a firm receives as a function of their current quality and level of investment. The transition function is restricted to increasing quality by one level, remaining at the current level, or decreasing quality by one level and the probability of each event is conditional on the level of investment, x.

In order to discuss numerical optimization methods for it is helpful to rewrite the right hand side of the Bellman equation as

$$\begin{aligned} & H(S) = \max_{\mu} f(S, \mu) \\ \text{s.t.} \quad & g(S, \mu) = 0, \\ & h(S, \mu) \geq 0 \end{aligned} \tag{2}$$

The most straight forward approach to solving this problem is to use a simple grid search. To do the grid search you choose a grid for each element in the vector μ and then evaluate f at each point in the grid and then choose the action that has the largest value. While this method will find the global maximum of f up to the accuracy limit imposed by how fine the grid is, this method requires far too many function evaluations to be used in many contexts.

The numerical analysis literature has produced many algorithms that have far better convergence rates to the optimal action and there are many efficiently implemented software packages that makes using these algorithms very easy. However, the run time required to find the optimal action varies significantly based on which method was chosen, the problem's structure, and the specific way that the algorithm was implemented in a software package.

The first numerical optimization algorithm I used for solving this problem was the method presented in Wachter and Biegler (2006) and implemented in the C++ library IPOPT. This method is a derivative free algorithm for solving constrained optimization problems and is also one of the most standard computational methods for solving non-linear optimization problems with constraints. It is also the default method for MATLAB's function fmincon.

When I used this algorithm for the numerical example, it had convergence issues and the software implementation was not suited to dynamic programming applications. This is because dynamic programming applications require the numerical method to be applied inside a for loop for the value function iterations and inside a for loop that is iterating over each approximation node. This means that the algorithm and software implementation should have limited overhead and not require access to a solver outside of the source code which the IPOPT library did.

Due to these limitations I then used the C++ library NLOPT since it does not have the implementation issues that IPOPT has (Johnson, 2017). This library provides access to many different optimization methods. The numerical methods to solve constrained non-linear optimization problems can be categorized into several categories. The first category, global optimization focuses on finding the global maximum. For example, the DIRECT algorithm presented in Jones, Perttunen, and Stuckman (1993) relies on the systematic division of the search domain into smaller and smaller hyperrectangles. Methods in this category are typically very slow and thus are not tractable for the applications of interest.

The second category is local derivative-free optimization algorithms. I compared NLOPT's implementation of the Constrained Optimization by Linear Approximations (COBYLA) presented in (M. Powell, 1998; M. J. Powell, 1994) to IPOPT's method. While methods in this category are not guaranteed to find the global optimum they are much faster and perform well on convex problems. I found that COBYLA was able to converge however the method required approximately 50 function evaluations per state per value function iteration to find the optimal action within a tolerance of 1e-5.

The third category is local gradient-based optimization algorithms. These methods require the analytical gradient of the optimization problem with respect to the decision variables to be supplied. This gradient is often easy to analytically derive and supply to the algorithm in the applications of interest. These methods have been shown to converge at a much faster rate than derivative-free algorithms (Bradie, 2006). I compared the L-BFGS algorithm presented in (Liu & Nocedal, 1989; Nocedal, 1980) with the COBYLA method and found that L-BFGS required approximately 5.6 function evaluations per state per value function iteration to find the optimal action within a tolerance of 1e-5. Local gradient-based

optimization algorithms also do not suffer from the curse of dimensionality in the action space which makes them attractive for applications where there are hundreds of decision variables and many constraints. This advancement in numerical analysis allows economic researchers to model firms with more realistic and complicated action spaces.

These methods all improve upon the simple grid search algorithm and depending on the context dramatically reduce f_{evals}. The derivative free methods however will still suffer from the curse of dimensionality in the action space that occurs when there are many decision variables. In order to overcome this issue it is recommended to analytically supply the gradient of f with respect to μ to the algorithm. If you supply the analytical gradient then it is possible to use algorithms in the local gradient-based optimization category which have been shown to further reduce f_{evals} and also overcome the curse of dimensionality in the action space.

4 Avoiding the curse of dimensionality in state space

The curse of dimensionality in the state space is often a more difficult issue to overcome. Typically, it is easier to solve computationally a model that is discrete rather than a continuous one and when the dimension of the state space is small and this is true in the Ericson and Pakes (1995) framework. However when the state space is large it is computationally more efficient to approximate the discrete state space with continuous functions.

There are many different methods for approximating a function evaluated at a few discrete points in one dimension. For example, if I was interested in approximating the function $v(S) = \sin(S)$ where $S \in [0, 4\pi]$. If I had the Lagrange data, $\{S_i, v_i\}_{i=1}^{N}$, where $S_i = \frac{4\pi i}{N}$, $v_i = \sin(S_i)$, and $N = 6$ then I could use many different interpolation methods. One method that I could used called polynomial interpolation with Chebyshev polynomials uses a sum of orthogonal polynomials to approximate the function v.

Chebyshev polynomials can be defined using a recursive definition or using a trionmetric definitive. Using the trigonometric definition, a n degree Chebyshev polynomial of the first kind is defined as

$$T_n(S) = \cos(n \arccos(\frac{2S - S_{min} - S_{max}}{S_{min} - S_{max}}))$$

If $S_{min} = -1$ and $S_{max} = 1$ then the first four polynomials will be

$$T_0(S) = 1$$
$$T_1(S) = S$$
$$T_2(S) = 2S^2 - 1$$
$$T_3(S) = 4S^3 - 3S$$
$$T_4(S) = 8S^4 - 8S^2 + 1$$

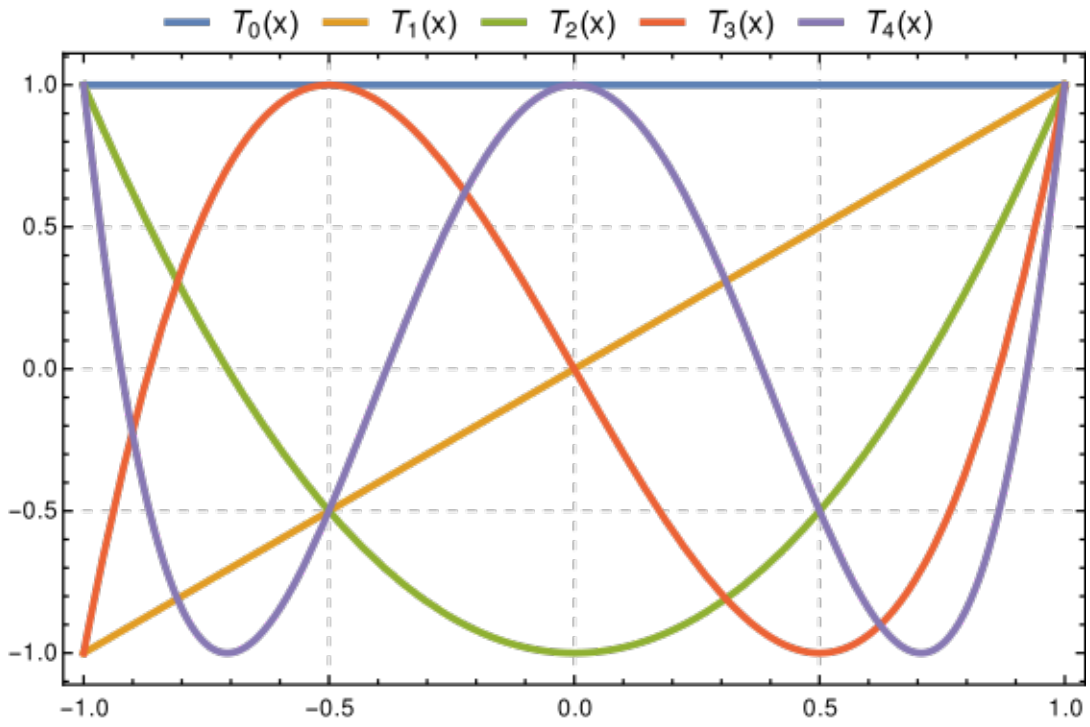

Figure 2: The first four Chebyshev polynomials

and then approximation can be written as $\hat{V}(S; b) = \sum_{k=0}^{n} b_k T_k(S)$. When interpolating a function with Chebyshev polynomials it is important to choose the approximation nodes based off of the roots of the polynomials. The Chebyshev nodes for one dimension are given by $S_i = (z_i + 1)(S_{max} - S_{min})/2 + S_{min}$ and $z_i = -\cos((2i-1)\pi/(2m))$ where m is the number

of approximation nodes. In order to approximate the function $v(S)$ using the Lagrange data $\{S_i, v_i\}_{i=1}^N$, where $v_i = v(S_i)$, it is necessary to solve for the coefficients, b, using

$$\min_{\boldsymbol{b}} \sum_{i=1}^{N} (\hat{V}(S_i; \boldsymbol{b}) - v_i)^2$$

This problem can be solved easily with OLS or with the Chebyshev Approximation Formula. This approximation method has a few cases where it performs poorly. One case where the approximation is poor is when N is too small. For a fixed N the approximation can be improved by fittiting the polynomial approximation using Hermite data. Hermite data consists of the approximation node, the function evaluated at the approximation node and also the gradient of the function evaluated at the approximation node. In order to fit the Chebyshev polynomial using Hermite data, $\{S_i, v_i, d_i\}_{i=1}^N$, where $d_i = \frac{dv(S_i)}{dS}$, the coefficients are found by solving the following OLS problem.

$$\min_{\boldsymbol{b}} \sum_{i=1}^{N} \left((\hat{V}(S_i; \boldsymbol{b}) - v_i)^2 + (\frac{\partial \hat{V}(S_i; \boldsymbol{b})}{\partial S} - d_i)^2 \right)$$

Chebyshev polynomials are an attractive choice for fitting a function using Hermite data since the derivative of type 1 Chebyshev polynomials are a function of type 2 Chebyshev polynomials. Since $\frac{\partial T_n(S)}{\partial S} = nU_{n-1}(S)$ and type 2 Chebyshev polynomials can be defined by the following equation.

$$U_n(S) = \frac{\sin((n+1)\arccos(\frac{2S - S_{min} - S_{max}}{S_{min} - S_{max}}))}{\sin(\arccos(\frac{2S - S_{min} - S_{max}}{S_{min} - S_{max}}))}$$

By incorporating the gradient information a function can be approximated to a given degree of accuracy with fewer approximation nodes. Figure 3 shows how Hermite approximation is able to approximate the function $\sin(x)$ with four approximation nodes to a much better degree of accuracy than the Lagrange approximation.

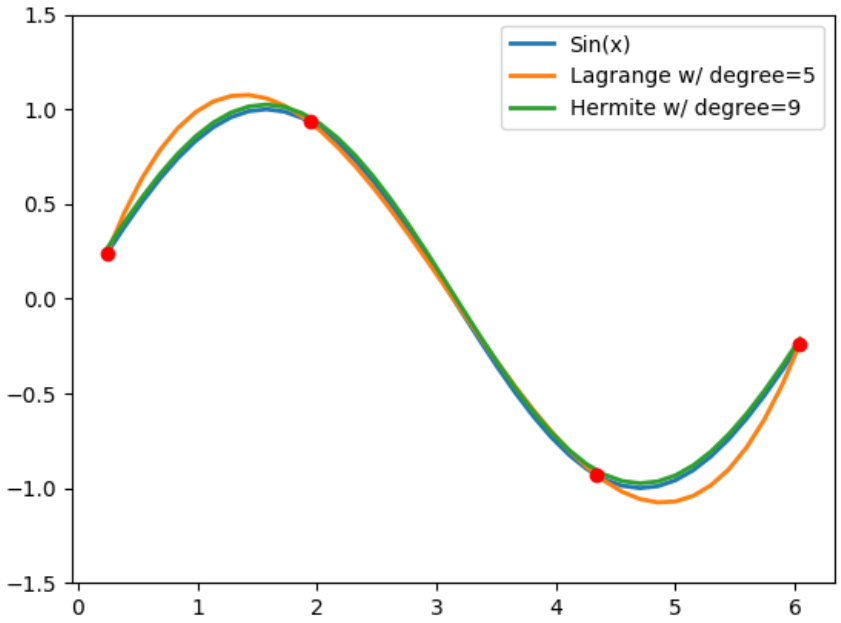

Figure 3: Lagrange and Hermite Chebyshev approximation of Sin(x)

Another case where polynomial approximations perform poorly is when there are large discontinuities or non-linearities in the function that is being approximated. When trying to approximate a discontinuous function polynomial approximations can often show the Gibbs phenomenon which is when the approximating polynomial overshooting or undershooting the function around a jump discontinuity. This overshooting/undershooting persists even when the number of approximation nodes increases. This can be seen in Figure 4 when Lagrange and Hermite Chebyshev polynomials are used to approximate the function $f(x)$ with 10 approximation nodes.

$$f(x) = \begin{cases} 1 & \text{if } x \leq 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

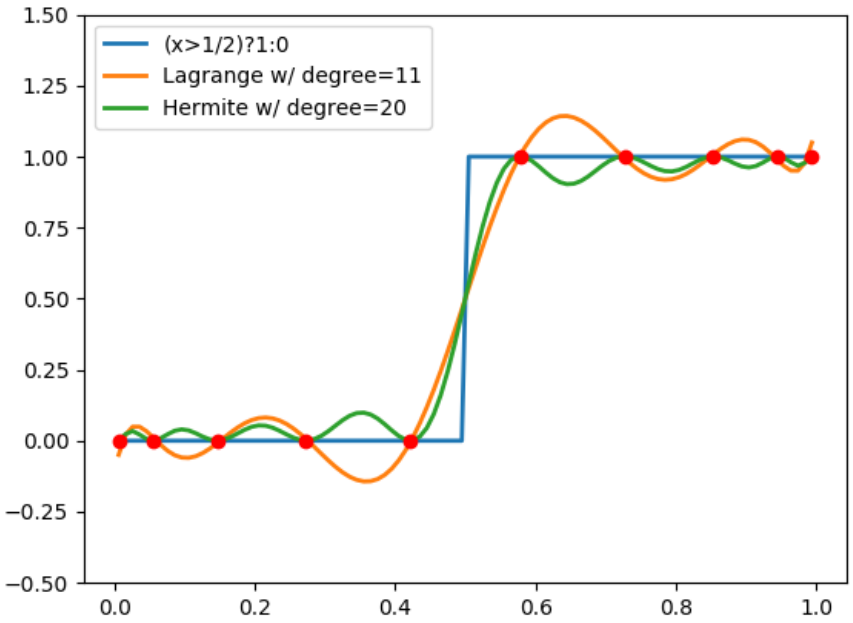

Figure 4: Lagrange and Hermite Chebyshev approximation of f(x)

In order to overcome this issue it is necessary to use an approximation techinque that is able to approximate the function locally. This can be done by fitting a piecewise linear function however, piece-wise linear functions are not smooth and the kink that occurs at each approximation node can cause problem when used for value function approximation. Cubic splines are smooth and can approximate the function locally however this method does not generalize to n dimensions.

Park and Sandberg (1991) show that Artificial Neural Nets are universal approximators. Artificial Neural Nets are able to approximate functions locally to avoid the Gibbs Phenomenon, and also scale very easily to approximating multidimensional functions. There has also been a large influx of computationally efficient software packages for training and evaluating neural nets. For an introduction to ANN please refer to Jain, Mao, and Mohiuddin (1996).

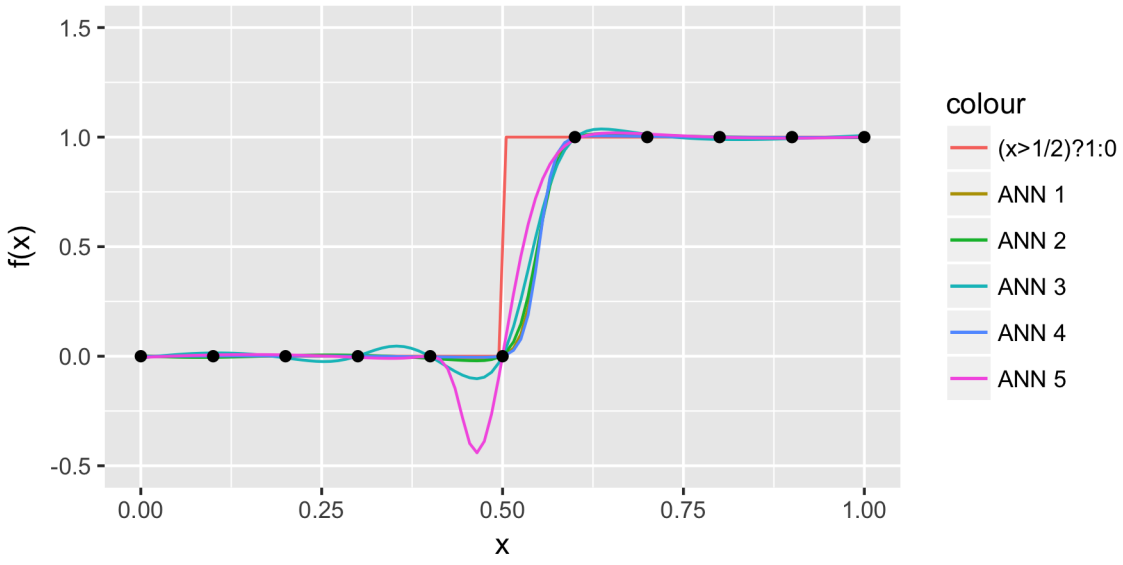

Figure 5: ANN approximation of f(x)

Figure 5 shows how a ANN with a single hidden layer with 10 nodes was able to approximate the function without the same issues that Chebyshev polynomials had. The graph was created using the same ANN but trained 5 times and each time starting out with a different set of random initial parameters. This figure also shows a drawback of using ANN, which is that they sometimes get stuck in local minimum. This can be seen with the fifth approximation. Zainuddin and Pauline (2008) showed that using different network structures such as a Radial Basis Function Network or a Wavelet Neural Network can improve the accuracy of ANN by reducing the frequency that the ANN gets stuck in a local minimum. Llanas, Lantarón, and Sáinz (2008) show that ANN can approximate any discontinuous function with a single hidden layer and Llanas and Lantarón (2007) also show that the Hermite approximation problem can be solved using ANN.

Using either Chebyshev polynomials or ANN to approximate a function allows for an accurate representation of a continuous function using a relatively small set of points. In the following algorithm these techniques will be used to approximate the value function of firm 1. By approximating the value function, it is possible to reduce the number of approximation nodes required to solve for an equilibrium.

Algorithm 2 Pakes McGuire Algo with L-VFA

procedure Choose N approximation nodes $\{S_i\}_{i=1}^N$, error tolerance for value function updating ϵ.

  Initialize $\boldsymbol{b}$

  while $\left\|\hat{V}_1^k - \hat{V}_1^{k-1}\right\| > \frac{\epsilon(1-\beta)}{2\beta}$ **do**

    for $i = 1, \ldots, N$ **do**

      Solve the Bellmann equation and store in memory $\hat{V}_{i,1}^k$ and $\mu_{i,1}^k$

$$v_{i,1}^k = \max_{\mu_{i,1}^k \in \Gamma(S_i)} \pi(S_i, \mu_{i,1}^k, \mu_{i,-1}^{k-1}) + \beta \mathbb{E}[\hat{V}(S'; b) | S_i, \mu_{i,1}^k, \mu_{i,-1}^{k-1})]$$

      Minimize the mean squared error and store in memory $\boldsymbol{b}$

$$\min_{\boldsymbol{b} \in \Omega} \sum_{i=1}^{N} \left(v_{i,1}^k - \hat{V}(S_i; \boldsymbol{b}) \right)^2$$

      Let $k = k + 1$

    end for

  end while

end procedure

This algorithm replaces the discretized value function with a continuous approximation. Since the value function can now be approximated by a continuous and differentiable function it is now possible to solve for an equilibrium of a model that uses a continuous state space. This is attractive feature since empirically the return from investment is continuously distributed, and since firms do not have a set of predetermined discrete levels of quality and so VFA allows for a more realistic theoretical model in addition to a more computationally efficient method for solving for an equilibrium.

Using the numerical example in this paper I compared the equilibrium found using two methods on six different grids of approximation nodes. The first method was using multidimensional Chebyshev polynomials to approximate the value function and the second method was a ANN with a single layer. When using the Chebyshev polynomials I used a grid based on the location Chebyshev nodes which are defined by $x_i = (z_i+1)(x_{max}-x_{min})/2+x_{min}$ and $z_i = -\cos((2i-1)\pi/(2m))$ where m is the number of approximation nodes per firm. When

using the ANN I used an equidistant gird of m points per firm. I examined the equilibrium value function and policy functions found when $m = 6, 12,$ and 18.

The results show that the ANN was able to approximate the value function to a high enough degree of accuracy that the equilibrium policy functions maintained their qualitative structure when $m = 6, 12,$ and 18. The multidimensional Chebyshev approximation was able to preserve the policy function's qualitative structure when $m = 12,$ and 18 however, when $m = 6$ the value function failed to converge. The approximation failed to converge because when the value function increases very quickly when firm 2's state is at a low value of about 3 and firm 1's state is between 3 and 6. This large rate of increase leads the approximation to exhibit the Gibbs Phenomenon which can be seen in Figure 7 and more prominently in Figure 8 [1]. This is especially an issue because during the value function iterations the region that was overshot is reinforced and the approximation error propagates through the subsequent value function iterations.

These results suggest that ANN are able to approximate the value function to a higher degree of accuracy than Chebyshev polynomials when the number of approximation nodes is reduced. This is useful since if Chebyshev polynomials can approximate it with a sufficient degree of accuracy when $m = 12$ and ANN can approximate the value function with a sufficient degree of accuracy when $m = 6$, then with 10 firms VFA with Chebyshev polynomials is approximately 60 times faster than a method that uses $m = 18$ and VFA with ANN is approximately 60,000 times faster than a method that uses $m = 18$.

L-VFA can also be easily combined with other methods for reducing the size of the state space. For example, it is easy to use parallelization to reduce the computational cost of the algorithm by finding the value of each state in parallel. This approximately divides the run time of the algorithm by the number of threads that evaluation of each state is parallelized over. In the numerical example I parallelized over 8 threads to reduce computation time.

It is also easy to incorporate other methods to reduce the size of the state space. Pakes and McGuire (1992) show that

$$V(\omega_1, \omega_2, \ldots, \omega_N) = V(\omega_1, \omega_{\pi(2)}, \ldots, \omega_{\pi(n)})$$

[1] My Github Repository has the source code, an easy to use makefile, and animated gifs of each of the 6 examples where each frame of the gif is a value function iteration.

for any $n-1$ dimensional vector $\pi = [\pi(2), \ldots, \pi(n)]$ which is a permutation of $(2, \ldots, n)$. Therefore, the value function does not need to be evaluated at each distinct approximation node but instead only needs to be evaluated at the nodes where $\omega_2 \geq \omega_3, \ldots, \geq \omega_n$. This causes the number of approximation nodes in the state space to be calculated as $\frac{(m+n-1)!}{(m-1)!n!}$ instead of m^n. For 10 firms and $m = 18$ this leads to $N = 8,436,285$, when $m = 12$ this leads to $N = 352,716$ and when $m = 6$ this leads to $N = 3,003$. Thus if it takes approximately 0.01 seconds to evaluate the function f, it will now only take 23.4 hours to do one value function iteration when $m = 18$, 0.98 hours to do one value function iteration when $m = 12$ and only 0.008 hours to do one value function iteration when $m = 6$. This reduction in computation time allows for the exploration of richer models with larger state spaces which were previously unable to be analyzed due to computational constraints.

Similar to how the approximation accuracy of one dimensional functions can be improved by fitting Hermite data rather than Lagrange data, the value function can be approximated using Hermite data. Cai and Judd (2015) discuss a method for easily obtaining the gradient of the value function in order use Hermite data to approximate the value function. Cai and Judd (2015) showed that Hermite value function approximation (H-VFA), was able to either produce a higher accuracy in a given amount of time or attain the same accuracy with much less computation time when compared to L-VFA.

If the right hand side of the Bellman equation is written as

$$\begin{aligned} & H(S) = \max_{\mu} f(S, \mu) \\ \text{s.t.} \quad & g(S, \mu) = 0, \\ & h(S, \mu) \geq 0 \end{aligned} \tag{4}$$

then the gradient of $H(S)$ is provided by

$$\frac{\partial H(S)}{\partial S_j} = \frac{\partial f}{\partial S_j}(S, \mu^*(x)) + \lambda_1^*(S)^T \frac{\partial g}{\partial S_j}(S, \mu^*(S)) + \lambda_2^*(S)^T \frac{\partial h}{\partial S_j}(S, \mu^*(S))$$

For $j = 1, \ldots, n$ and n is the dimension of S. The computation of the gradient requires finding several derivatives and is difficult to compute. However, the problem can be rewritten by adding in the variable y such that

$$
\begin{aligned}
& H(S) = \max_{\mu, y} f(y, \mu) \\
\text{s.t.} \quad & g(y, \mu) = 0, \\
& h(y, \mu) \geq 0, \\
& S_j - y_j = 0, \quad j = 1, \ldots, n.
\end{aligned}
\tag{5}
$$

Then the gradient of $H(S)$ is provided by

$$
\frac{\partial H(S)}{\partial S_j} = \tau_j^*(S)
$$

where $\tau_j^*(S)$ is the Lagrange multiplier for the constraints $S_j - y_j = 0$.

There are several derivative-free constrained optimization software libraries that provide along with the solution to the maximization problem the Lagrange multiplier for each constraint. This method allows for an easy method to obtain the gradient of the value function and thus able to use the Hermite data, $\{S_i, v_i, d_i\}_{i=1}^N$, where d_i is the gradient vector evaluated at S_i, to approximate the value function.

The coefficients to approximate the value function using multidimensional Chebyshev polynomials are obtained by solving the following equation and the notation for multidimensional Chebyshev polynomials is presented in Appendix 2.

$$
\min_b \left\{ \sum_{i=1}^N \left(v_i - \sum_{0 \leq |\alpha| \leq r} b_\alpha T_\alpha(S_i) \right)^2 + \sum_{i=1}^N \sum_{j=1}^n \left(d_{i,j} - \sum_{0 \leq |\alpha| \leq r} b_\alpha \frac{\partial}{\partial S_j} T_\alpha(S_i) \right)^2 \right\}
$$

While the L-VFA algorithm presented in Cai and Judd (2015) is easily added to the Pakes McGuire Algorithm, the H-VFA algorithm is not directly applicable. A direct adaptation of H-VFI for the Pakes McGuire Algorithm would be

Algorithm 3 Pakes McGuire Algo with H-VFA using Chebyshev Polynomials

procedure Choose N approximation nodes $\{S_i\}_{i=1}^N$, error tolerance for value function updating ϵ.

  Initialize $\boldsymbol{b}$

  while $\left\|\hat{V}_1^k - \hat{V}_1^{k-1}\right\| > \frac{\epsilon(1-\beta)}{2\beta}$ **do**

    for $i = 1, \ldots, N$ **do**

      Obtain the value function at S_i, and gradient by solving Equation (4) or Equation (5). Then store in memory $v_{i,1}^k$, $\mu_{i,1}^k$, and d_i^k.

      Minimize the mean squared error and store in memory b using Hermite data $\{S_i, v_{i,1}, d_{i,1}\}_{i=1}^N$ by solving

$$\min_{\boldsymbol{b}} \left\{ \sum_{i=1}^{N} \left(v_i - \sum_{0 \leq |\alpha| \leq r} \boldsymbol{b}_\alpha T_\alpha(S_i) \right)^2 + \sum_{i=1}^{N} \sum_{j=1}^{n} \left(d_{i,j} - \sum_{0 \leq |\alpha| \leq r} \boldsymbol{b}_\alpha \frac{\partial}{\partial S_j} T_\alpha(S_i) \right)^2 \right\}$$

      Let $k = k + 1$

    end for

  end while

end procedure

Step 5: in this algorithm causes complications. Since the reaction of other firms is taken into account a firm's decision the gradient calculation using (4) would require the calculation of the gradient of the policy function. This is because the policy that the other firms choose depends on what the industry's state is. Thus in order complete Step 5: using Equation (4) it is necessary to determine the gradient of the policy function. Since the analytical gradient of the policy function cannot be obtained it is necessary to numerically approximate the gradient. This can be done easily with the central difference method. The accuracy of the central difference method is limited by the step size which in this context is the distance between two approximation nodes. Thus in order to obtain an accurate approximation of the policy function there would need to be more approximation nodes which prevents H-VFA from decreasing the number of approximation nodes.

If Step 5: is calculated using (5) then there can be an issue with the numerical optimization method converging. This is due to the fact that a derivative free algorithm is used to

calculate the solution and that by adding in an additional decision variable y the number of function evaluations required to find the optimal action will grow significantly. Therefore the computational cost of finding the optimal action may cause the Pakes Mcguire Algorithm with H-VFA to be slower than Pakes Mcguire Algorithm with L-VFA.

5 Conclusion

I have shown how to incorporate numerical optimization methods to avoid the curse of dimensionality in the action space and that by adding value function approximation to the Pakes McGuire algorithm it is possible to reduce the number of states needed per firm. I have also discussed the benefits that ANN have over multidimensional Chebyshev polynomials for value function approximation and the complications that arise from attempting to fit the value function using Hermite data in dynamic game models. These conclusions are supported by a numerical example of the Pakes McGuire quality ladder model.

The L-VFA numerical example when fit with ANN used a simple single hidden layer ANN. Future research could examine the benefits to using more complex network structure such as deep neural networks. Liang and Srikant (2016) show that the number of neurons needed to approximate a function decreases exponentially in the number of layers in the network. Future research could also incorporate other methods from approximate dynamic programming such as using post-decision state sampling to reduce the computational cost of computing the expected value, or using ANN to construct a continuous implementation of the dynamic lookup table presented in Ulmer, Mattfeld, and Köster (2017).

References

Aguirregabiria, V., & Nevo, A. (2013). Recent developments in empirical io: Dynamic demand and dynamic games.

Benkard, C. L., Jeziorski, P., & Weintraub, G. Y. (2015). Oblivious equilibrium for concentrated industries. *The RAND Journal of Economics*, *46*(4), 671–708.

Bradie, B. (2006). *A friendly introduction to numerical analysis: with c and matlab materials on website*. Prentice-Hall.

Cai, Y., & Judd, K. L. (2015). Dynamic programming with hermite approximation. *Mathematical Methods of Operations Research*, *81*(3), 245–267.

Chen, J., & Doraszelski, U. (2006). Network effects, compatibility choice, and industry dynamics: The duopoly case.

Chen, J., Doraszelski, U., & Harrington Jr, J. E. (2009). Avoiding market dominance: Product compatibility in markets with network effects. *The RAND Journal of Economics*, *40*(3), 455–485.

Doraszelski, U., & Judd, K. (2007). *Dynamic stochastic games with sequential state-to-state transitions* (Tech. Rep.). Working paper, Harvard University, Cambridge.

Doraszelski, U., & Judd, K. L. (2012). Avoiding the curse of dimensionality in dynamic stochastic games. *Quantitative Economics*, *3*(1), 53–93.

Doraszelski, U., & Pakes, A. (2007). A framework for applied dynamic analysis in io. *Handbook of industrial organization*, *3*, 1887–1966.

Doraszelski, U., & Satterthwaite, M. (2010). Computable markov-perfect industry dynamics. *The RAND Journal of Economics*, *41*(2), 215–243.

Ericson, R., & Pakes, A. (1995). Markov-perfect industry dynamics: A framework for empirical work. *The Review of Economic Studies*, *62*(1), 53–82.

Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, *29*(3), 31–44.

Johnson, S. G. (2017). The nlopt nonlinear-optimization package. Retrieved from http://ab-initio.mit.edu/wiki/index.php/NLopt (NLopt Version 2.4.2)

Jones, D. R., Perttunen, C. D., & Stuckman, B. E. (1993). Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, *79*(1), 157–181.

Liang, S., & Srikant, R. (2016). Why deep neural networks? *CoRR*, *abs/1610.04161*. Retrieved from http://arxiv.org/abs/1610.04161

Liu, D. C., & Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, *45*(1), 503–528.

Llanas, B., & Lantarón, S. (2007). Hermite interpolation by neural networks. *Applied Mathematics and Computation*, *191*(2), 429–439.

Llanas, B., Lantarón, S., & Sáinz, F. J. (2008). Constructive approximation of discontinuous functions by neural networks. *Neural Processing Letters*, *27*(3), 209–226.

Mitchell, M. F., & Skrzypacz, A. (2006). Network externalities and long-run market shares. *Economic Theory*, *29*(3), 621–648.

Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Mathematics of computation*, *35*(151), 773–782.

Pakes, A., & McGuire, P. (1992). *Computing markov perfect nash equilibria: Numerical implications of a dynamic differentiated product model.* National Bureau of Economic Research Cambridge, Mass., USA.

Pakes, A., & McGuire, P. (2001). Stochastic algorithms, symmetric markov perfect equilibrium, and the curse of dimensionality. *Econometrica*, *69*(5), 1261–1281.

Park, J., & Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. *Neural computation*, *3*(2), 246–257.

Powell, M. (1998). Direct search algorithms for optimization calculations. *Acta numerica*, *7*, 287–336.

Powell, M. J. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis* (pp. 51–67). Springer.

Turner, R. (2017). deldir: Delaunay triangulation and dirichlet (voronoi) tessellation [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=deldir (R package version 0.1-14)

Ulmer, M. W., Mattfeld, D. C., & Köster, F. (2017). Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*.

Wachter, A., & Biegler, L. T. (2006). On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57.

Weintraub, G. Y., Benkard, C. L., & Van Roy, B. (2008). Markov perfect industry dynamics with many firms. *Econometrica*, *76*(6), 1375–1411.

Zainuddin, Z., & Pauline, O. (2008). Function approximation using artificial neural networks. *WSEAS Transactions on Mathematics*, *7*(6), 333–338.

Appendix 1: Plots

In each plot the top row is the value function, the middle row is the policy function for investment, and the bottom row is the policy function for product price. Each column shows a different view of the 3D graph where the first column shows firm 2's quality increasing on the left and firm 1's quality increasing on the right. The middle column shows a rotation of the first column so that firm 1's quality is increasing to the right and the right column the rotation such that firm 1's quality is decreasing to the right. The top row was generated by evaluating the value function approximation on a grid of 150x150 points, and coloring the points based on their value. The middle and bottom rows cannot be evaluated in this way since the algorithm presented do not interpolate the policy functions. Since it is hard to see the structure of the policy functions when only examining discrete points I interpolated the policy functions using a Delaunay triangulation method from the package deldir in R (Turner, 2017).

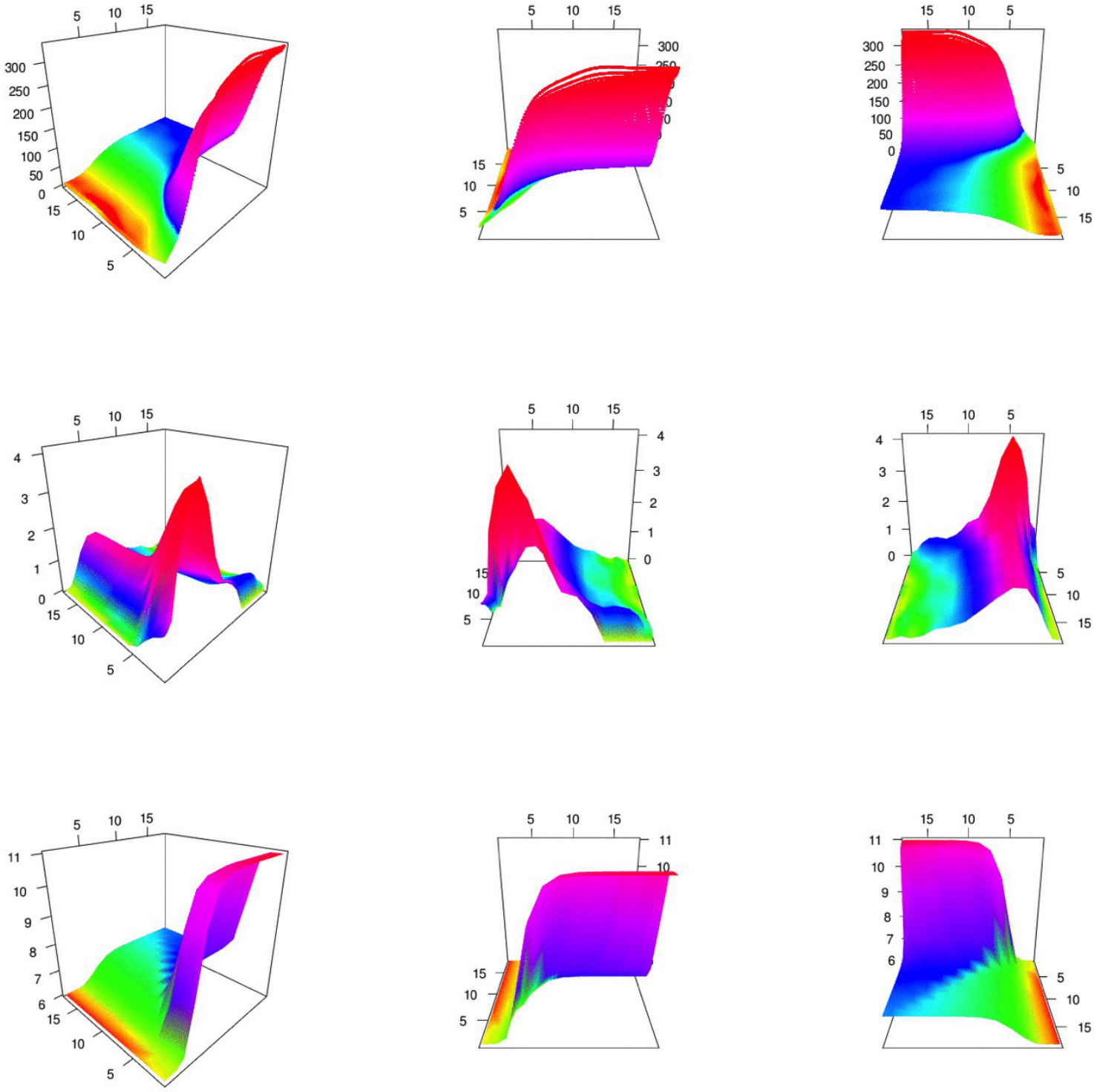

Figure 6: The equilibrium value, investment policy and pricing policy fit on an 18x18 grid with 12 degree multidimensional Chebyshev polynomials

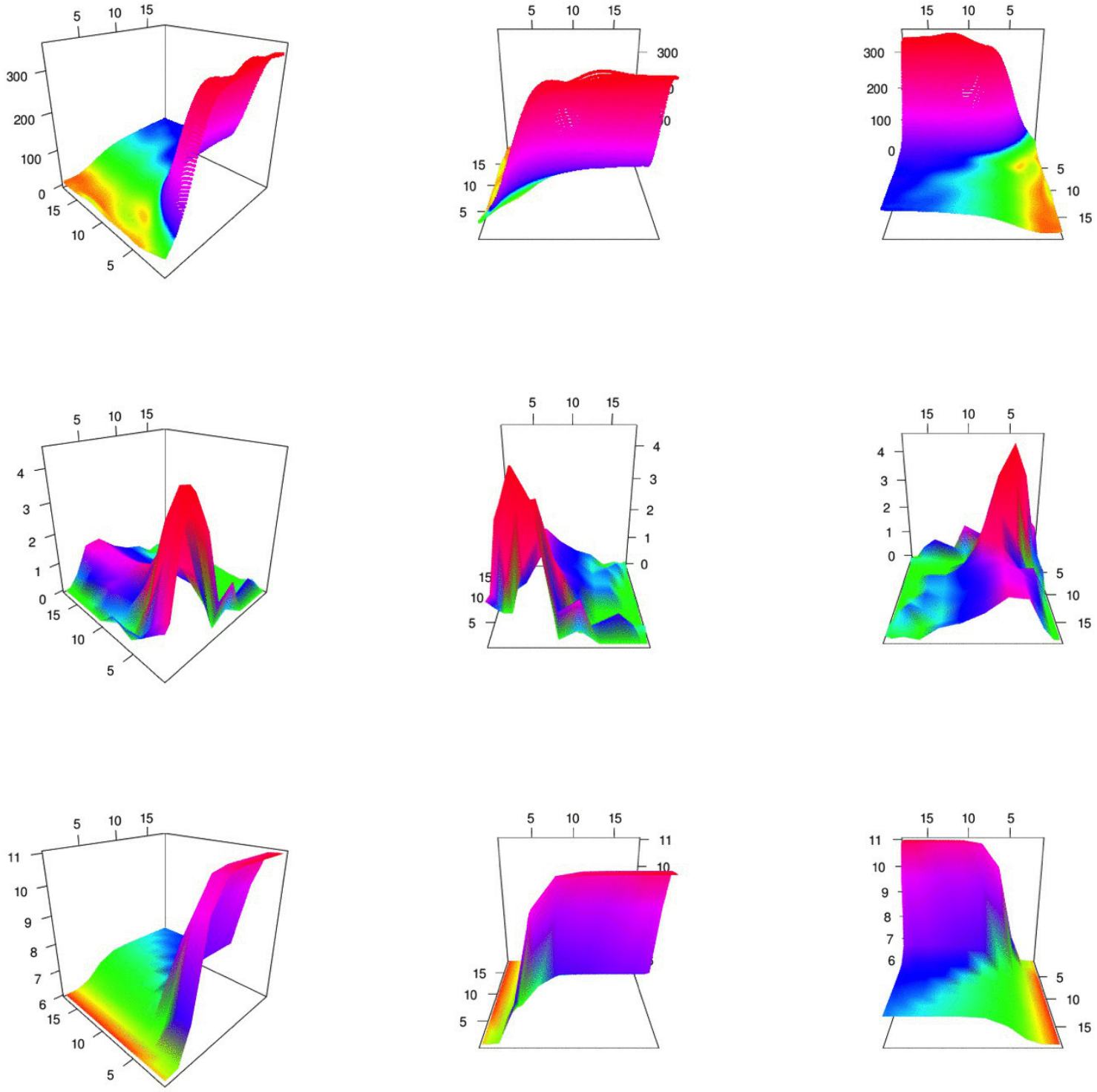

Figure 7: The equilibrium value, investment policy and pricing policy fit on an 12x12 grid with 12 degree multidimensional Chebyshev polynomials

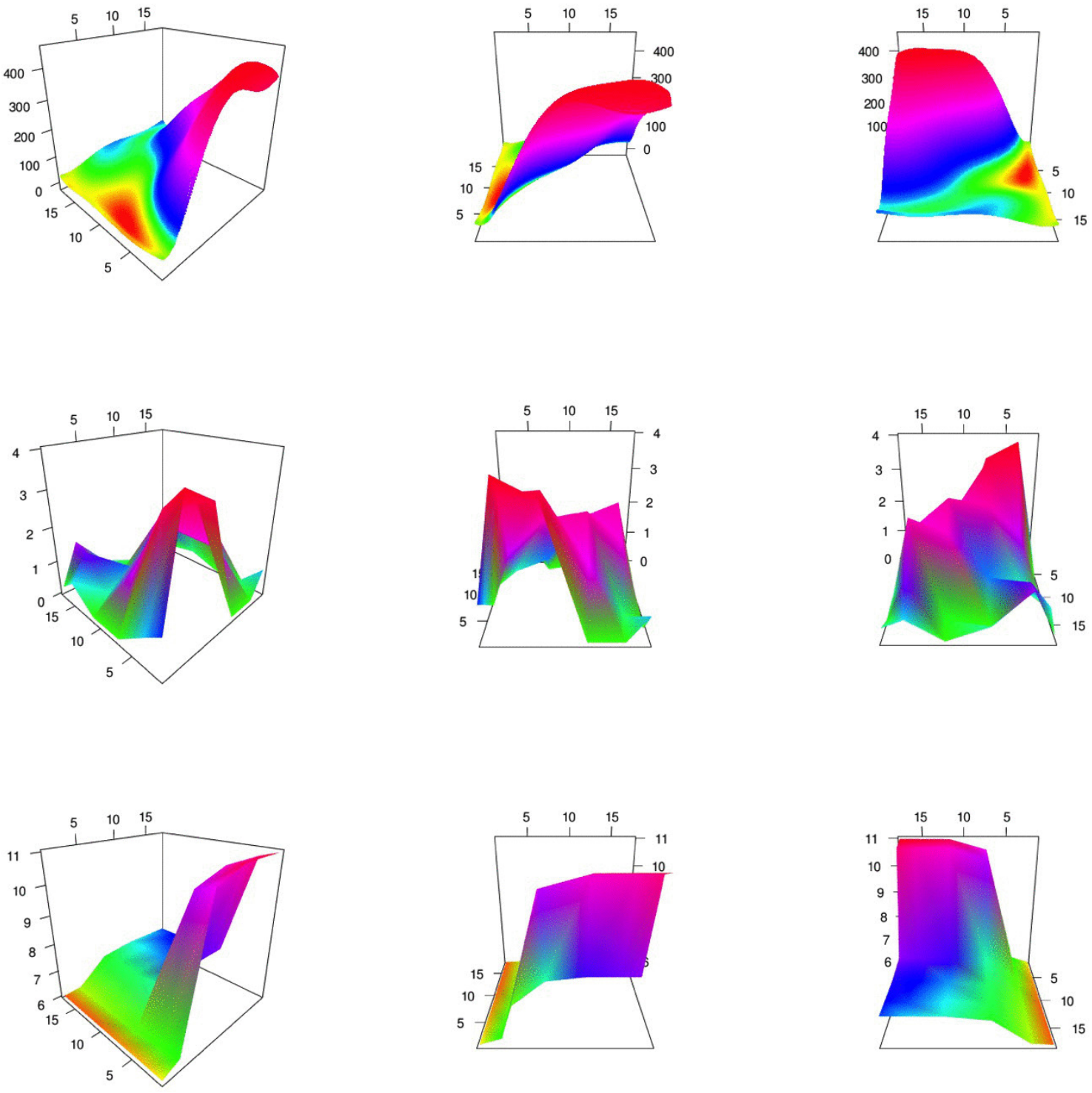

Figure 8: The equilibrium value, investment policy and pricing policy fit on an 6x6 grid with 6 degree multidimensional Chebyshev polynomials

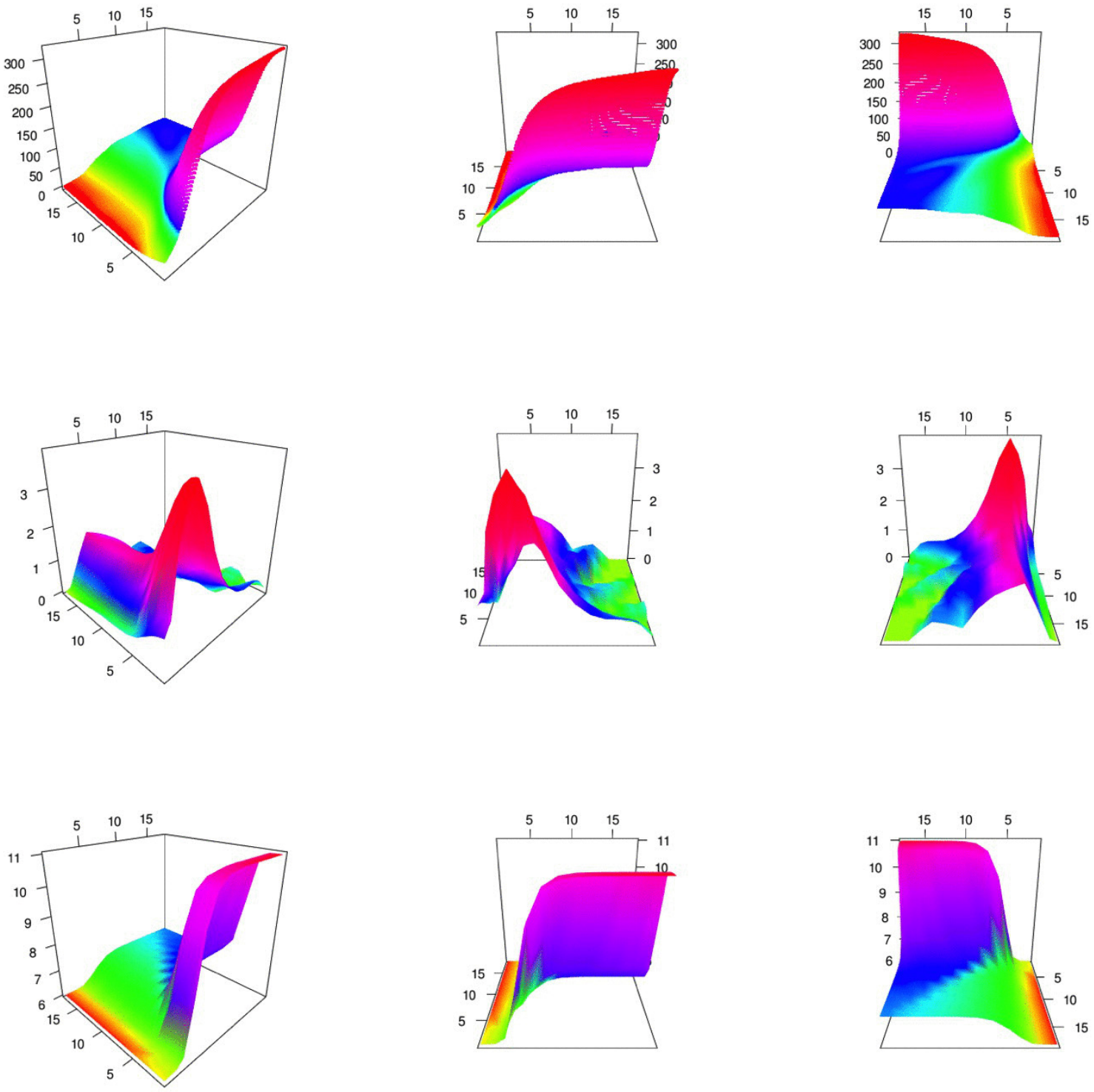

Figure 9: The equilibrium value, investment policy and pricing policy fit on an 18x18 grid using a neural net with a single fully connected layer with 324 nodes and tanh activation function

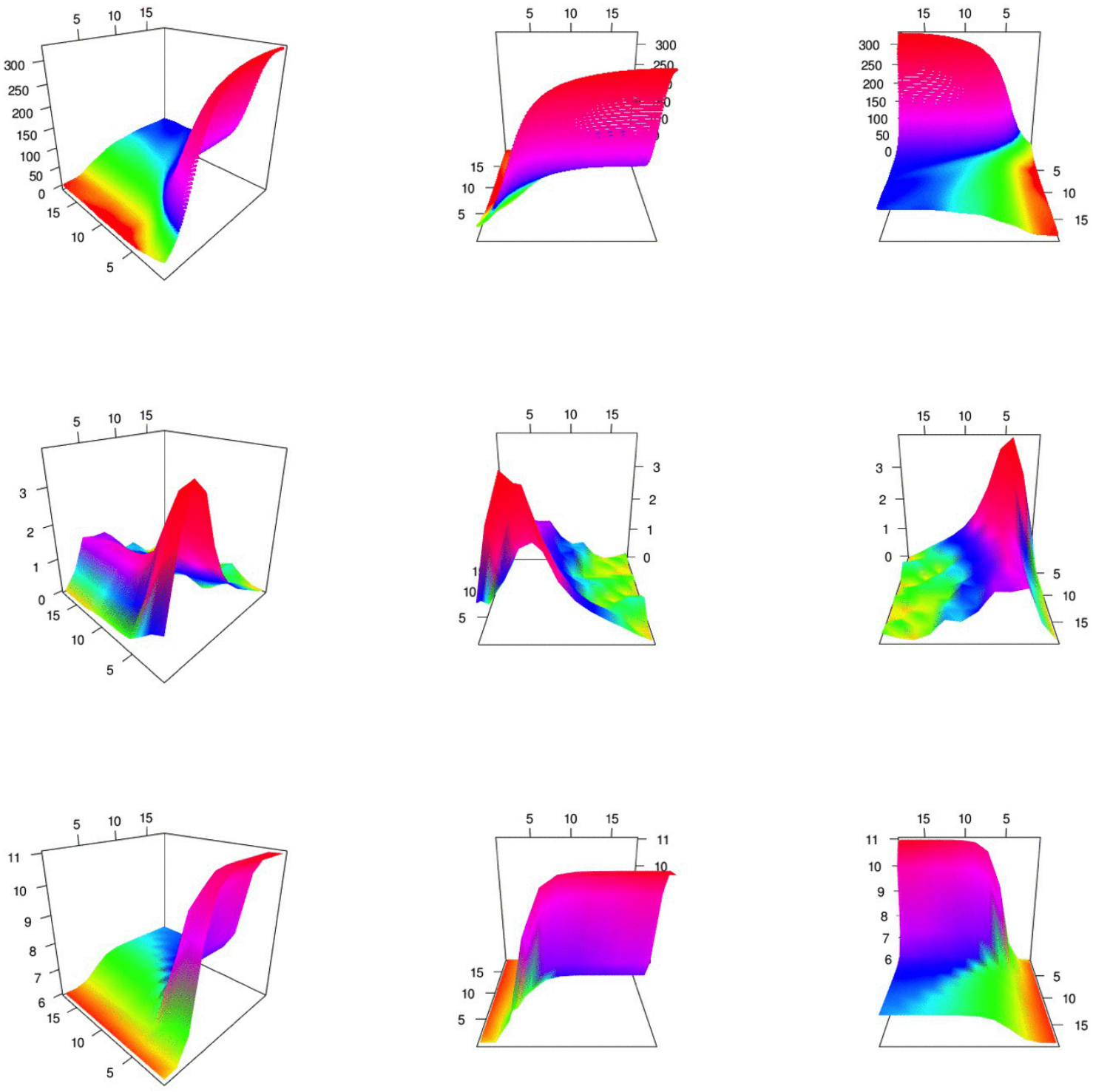

Figure 10: The equilibrium value, investment policy and pricing policy fit on an 12x12 grid using a neural net with a single fully connected layer with 144 nodes and tanh activation function

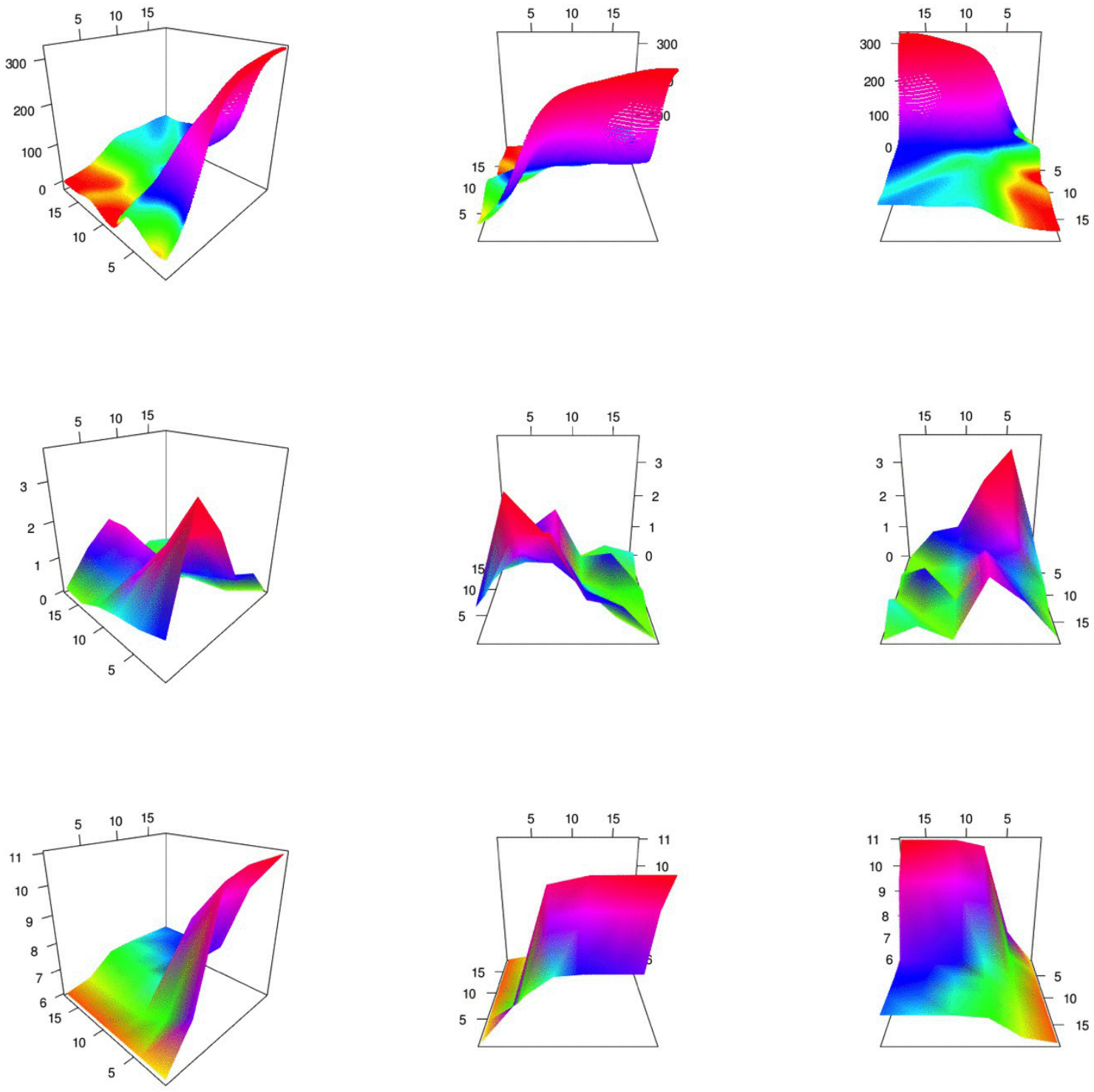

Figure 11: The equilibrium value, investment policy and pricing policy fit on an 6x6 grid using a neural net with a single fully connected layer with 144 nodes and tanh activation function

Appendix 2: Multidimensional complete Chebyshev polynomial approximation

This section is reproduced from Cai and Judd (2015) for convenience. Let the domain for the approximation be

$$\{\boldsymbol{x} = (x_1, \ldots, x_d) : x_{min,j} \leq x_j \leq x_{max,j}, j = 1, \ldots, d\},$$

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

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

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

Using this notation, the degree n complete Chebyshev approximation for $V(\boldsymbol{x})$ is

$$\hat{V}_n(\boldsymbol{x}; \boldsymbol{b}) = \sum_{0 \leq |\alpha| \leq n} b_\alpha T_\alpha(\boldsymbol{Z}(\boldsymbol{x})),$$

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