

# **Approximating and Simulating the Stochastic Growth Model: Parameterized Expectations, Neural Networks, and the Genetic Algorithm**

John Duffy  
Department of Economics  
University of Pittsburgh  
Pittsburgh, PA 15260, USA  
jduffy@pitt.edu

Paul D. McNelis  
Department of Economics  
Georgetown University  
Washington, DC 20057, USA  
mcnelisp@gunet.georgetown.edu

This Draft: November 1999

## **Abstract**

This paper suggests a new approach to solving the one-sector stochastic growth model using the method of parameterized expectations. The approach is to employ a "global" genetic algorithm search for the parameters of the expectation function followed by a "local" gradient-descent optimization method to ensure fine-tuning of the approximated solution. We use this search procedure in combination with either polynomial or neural network specifications for the expectation function. We find that our approach yields highly accurate solutions in the case where an exact analytic solution exists as well as in cases where no closed-form solution exists. Our results further suggest that neural network specifications for the expectation function may be preferred to the more commonly used polynomial specification.

*JEL classification:* C63; O41

*Key words:* Solution methods; stochastic growth model; parameterized expectations; genetic algorithm; neural networks.

## 1. Introduction

This paper compares alternative methods for approximating and solving the stochastic growth using the method of parameterized expectations. We distinguish between polynomial and neural network specifications for expectations and between gradient-descent and genetic algorithm methods for solving these models of parameterized expectations.

Parameterized expectation methods, developed by Marcet (1988) and Den Haan and Marcet (1990, 1994), work on the Euler equation that characterizes the solution to the stochastic growth model. This approach involves approximation of the conditional expectation function in the Euler equation with a pre-specified functional form and the use of an optimization method to determine the parameterization of this functional form.<sup>1</sup>

Most approaches to parameterized expectations rely on a polynomial approximation for the expectations mechanism and some type of iterative, gradient-descent-based method for finding the appropriate parameter values of the expectations function. Den Haan and Marcet (1990, 1994), for example, use non-linear least squares, while Christiano and Fisher (1997) use ordinary least squares in their modified parameterized expectations approach. We propose the use of a neural network specification for the expectations function and the use of a real-coded genetic algorithm to obtain the parameter values for the neural network specification.

Other researchers have also proposed the use of such artificial intelligence techniques as an aid to solving stochastic growth models. Beaumont and Bradshaw (1995, 1996) use distributed parallel genetic algorithms, Chyi (1997) uses neural networks, and Schmertmann (1996) uses genetic programming. However, there has been little effort to combine or hybridize these techniques. Moreover, these researchers have used these techniques for direct recovery of the *policy functions* that solve the Euler

---

<sup>1</sup> Thus, the method of parameterized expectations belongs to the more general class of projection methods for solving functional equations as discussed in Judd (1992, 1998).

equation. As Christiano and Fisher (1997) point out, a direct search for policy functions can be difficult, especially in cases where there may be binding constraints, since the researcher has to approximate the policy functions while simultaneously satisfying all Kuhn-Tucker conditions.

By contrast, the method of parameterized expectations is easier to implement, as it applies directly to the Euler equation so that the Kuhn-Tucker conditions are automatically satisfied, and the resulting approximation of the expectation function can be used to recover the policy functions.

For approximation purposes, neural network specifications have been shown to be more efficient than polynomial specifications as they avoid the need for exponentially increasing numbers of cross product terms. Thus, a neural network specification delivers greater precision for the same number of parameters, or equal precision with fewer parameters.

The search for the parameterization of a polynomial approximation in the standard parameterized expectations approach is typically conducted using gradient-descent identification methods that require arbitrary starting conditions for the initial parameters. Depending on the choice of these initial parameters, the solution process may fail to converge or may converge to local rather than global optima.

The genetic algorithm-based search for optimal parameter values that we employ differs markedly from these gradient-descent methods and offers some advantages with respect to the problem of choosing initial conditions. One difference is that genetic algorithms avoid the need to take derivatives, thus allowing for the possibility of a much larger class of functional approximations.

The genetic algorithm also differs from gradient descent methods in that it is a *population-based*, evolutionary search process that begins with a very large set of initial candidate parameter vectors. These vectors are subjected to selection pressure based on relative fitness and other genetic operators that serve to advance the search for

increasingly fit parameter vectors, where the fitness metric is specified by the user. In the case of the growth model, this metric is minimization of the Euler equation residuals.

Genetic algorithms have been shown to optimize on the trade-off between experimenting with new candidate solutions and utilizing solutions that have worked well in the past (Holland (1975)). Because these search algorithms are population-based, they are very well suited to searching over large parameter spaces and are especially useful in situations where the choice of good initial parameter values may not be well known. As Dorsey and Mayer (1995) point out, while the genetic algorithm may not find the point at which the gradient is exactly zero, it will often come very close to achieving this goal. Genetic algorithms thus can greatly enhance the performance of “locally efficient gradient-type algorithms” by providing the “needed global efficiency” [Dorsey and Mayer (1995): p. 565].<sup>2</sup>

A drawback, of course, is that the genetic algorithm is computationally more time consuming as compared with other methods, such as the standard gradient descent approach.

The purpose of this paper is to show that a neural network approximation in combination with a genetic algorithm search may give more accurate results than the more commonly implemented parameterized expectations methods based on polynomial approximations and gradient-descent optimization. Of course, computational time considerations may prevent the genetic algorithm from taking the place of faster methods. However, our results suggest that researchers working with relatively noisy or highly nonlinear models might want to compare the results they obtain using parameterized expectations with polynomial approximations, as well as other solution methods, with the results they obtain using neural network approximations and genetic algorithms.

---

<sup>2</sup> Dorsey and Mayer (1995) analyzed the performance of the GA for static non-linear econometric estimation. Our work can be viewed as an extension of their work. We analyze the performance of the GA for the solution of dynamic non-linear optimization models, rather than for estimation purposes.

In the next section we provide a description of our alternative neural network/genetic algorithm approach for solving the benchmark stochastic growth model as described and analyzed by Taylor and Uhlig (1990), Den Haan and Marcet (1990, 1994) and many others. We also present a brief review of the parameterized expectations approach. In section 3 we report and discuss our simulation results for two versions of the model, one in which an analytic solution for the model is available and another version in which there is no closed-form solution. Finally, section 4 provides a summary and some concluding remarks.

## 2. Two Approaches to approximating and solving the Stochastic Growth Model Using Parameterized Expectations

Consider the following one-sector stochastic growth model:

$$\begin{aligned}
 & \text{Maximize} \\
 & E_0 \left[ \sum_{t=0}^{\infty} \beta^t U(c_t) \right] = E_0 \left[ \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-t}}{1-t} \right] \\
 & \text{subject to} \\
 & c_t + k_t - k_{t-1} = q_t k_{t-1}^a - d k_{t-1} \quad (1) \\
 & \text{where} \\
 & \ln(q_t) = r \ln(q_{t-1}) + e_t, \\
 & e_t \sim N(0, \sigma^2), \\
 & \beta > 0, 0 < a < 1, \\
 & c_t > 0, k_t > 0, \forall t, k_{-1} \text{ given.}
 \end{aligned}$$

Here  $c_t$  denotes consumption at time  $t$ ,  $k_t$  denotes the capital stock at time  $t$ ,  $\beta \in (0, 1)$  denotes the constant discount factor,  $d$  is the constant rate at which the capital stock depreciates from  $t-1$  to  $t$ , and  $q_t$  denotes the time  $t$  stochastic shock to technology that evolves according to an autoregressive process with parameter  $r \in (-1, 1)$  and an error process  $e_t \sim N(0, \sigma^2)$ .

The solution to this optimization problem yields a policy function:  
and a law of motion for the stock of capital:

$$c_t = h(k_t, q_t),$$

$$k_{t+1} = g(k_t, q_t) = q_t k_t^a + (1-d)k_t - h(k_t, q_t).$$

It can be shown that equation system (1) has a closed form analytical solution only when  $\tau=\delta=1$  (the case of logarithmic preferences and full depreciation of the capital stock in every period).<sup>3</sup>

In this special case, the optimal policy function and path for the capital stock are given by the following system:

$$\begin{aligned} c_t &= (1-ab)q_t k_t^a, \\ k_{t+1} &= abq_t k_t^a. \end{aligned}$$

For all other cases that researchers have examined, numerical approximation methods are needed to solve for the optimal policy function and the capital accumulation path, given the stochastic shock process and the constraints in (1).

An alternative to direct approximation of the policy functions is to use the method of parameterized expectations due to Marcet (1988) and Den Haan and Marcet (1990, 1994). This method works on the Euler equation that emerges from the first order necessary conditions to the maximization problem:

$$c_t^{-1} = bE_t[c_{t+1}^{-1}(q_{t+1}k_t^{a-1}a + 1-d)]. \quad (2)$$

The method of parameterized expectations proceeds in the following sequence of steps. First, substitute the conditional expectation function on the right-hand side of the Euler equation (2) by a parameterized function,  $\Psi(g; k_{t-1}, q_t)$ , in order to obtain

$$c_t^{-1} = b\Psi(g; k_{t-1}, q_t), \quad (3)$$

where  $g$  denotes a vector of parameters.

Then generate a single draw of a series for  $q_t$  of length  $T$ . This series is drawn just once and is used repeatedly in finding a solution.

---

<sup>3</sup> See, e.g., Sargent (1987) p. 122.

Next, create a series for  $\{c, k\}$  of length  $T$  using equation (3) for  $c_t$  and then obtaining values for  $k_t$  via the budget constraint in (1).

Finally, run the non-linear least squares regression

$$l_{t+1} = y(g; k_{t-1}, q_t), \quad (4)$$

where  $l_{t+1} = [c_{t+1}^{-1} (q_{t+1} k_t^{a-1} a + 1 - d)]$ . This regression iterates until the parameter vector  $\gamma$  minimizes the sum of squared differences between  $l_{t+1}$  and the parameterized function  $\psi$ , to some degree of precision (Marcet and DenHaan (1990) use four digits of accuracy).

A common way to parameterize equation (3) is using a polynomial approximation. Den Haan and Marcet (1994) showed that the second-order, logarithmic polynomial expansion, involving six parameters,

$$l_{t+1} = g_0 + g_1 \ln(k_{t-1}) + g_2 \ln(q_t) + g_3 \ln(k_{t-1})^2 + g_4 \ln(k_{t-1}) \ln(q_t) + g_5 \ln(q_t)^2 \quad (5)$$

provides a reasonably accurate approximation to the solution to the one--sector stochastic growth model. We work with this same polynomial approximation as one of the ways in which we implement the method of parameterized expectations.

An alternative to a polynomial approximation is to use a neural network approximation. While neural networks were originally developed as models of the workings of the brain, they can also be used as alternatives to polynomials as function approximators. The neural network approximation we chose to consider uses a logsigmoid expansion and is written as:

$$\begin{aligned} l_{t+1} &= g_0 + g_1 \Omega_t^1 + \Omega_t^2, \\ w_t^1 &= g_2 k_{t-1} + g_3 q_t, \\ w_t^2 &= g_4 k_{t-1} + g_5 q_t, \\ \Omega_t^i &= \frac{1}{1 + e^{-w_t^i}}, i = 1, 2. \end{aligned} \quad (6)$$

Equation (6) is referred to as a *single hidden layer logsigmoid* neural network with two hidden nodes or "neurons,"  $\Omega_i^i, i=1,2$ .<sup>4</sup>

Hornik, Stinchcombe and White (1989) have proved that with sufficiently many neurons, single hidden layer sigmoidal neural networks (i.e. linear combinations of sigmoidal functions) can approximate any arbitrary function arbitrarily well in the space of useful functions. For comparison purposes, our neural network approximation (6) involves the *same* number of parameters -- six -- as the alternative, second-order polynomial approximation that we also consider.

The logsigmoid function used in equation (6) is one of the most commonly used activation functions in neural network approximations. As Sargent (1993) has pointed out, single hidden layer sigmoidal networks can achieve the same function approximation accuracy using fewer parameters than traditional polynomial expansions. In particular, Barron (1993) has proved that single hidden layer sigmoidal networks can achieve lower approximation error rates using only linearly increasing numbers of parameters; polynomial expansions can achieve these same approximation error rates only by using exponentially increasing numbers of parameters.

In neural network estimation, the coefficient vector,  $g$ , is typically obtained through the backpropagation method, an iterative non-linear process similar to non-linear regression. As in most gradient-descent methods, a random set of coefficients is used to initialize the "learning," and the coefficient vector is gradually adjusted through search methods based on the first and second derivatives of an "error function." Marcet and Den Haan's (1990, 1994) parameterized expectation method operates in a similar manner on the coefficient vector of the polynomial expansion, making use of a simple iterative procedure to update this coefficient vector.

---

<sup>4</sup> See Bishop (1995) for an up-to-date treatment of neural network methods, and Kuan and White (1994) for a clear discussion of the econometrics of neural networks.

An alternative optimization method is to use a *genetic algorithm*. A genetic algorithm is a stochastic, directed search algorithm that has proved useful in finding global optima in both static and dynamic environments.

Genetic algorithms work with large populations of candidate solutions that are repeatedly subjected to selection pressure (survival of the fittest) and which undergo naturally occurring genetic operations in the search for improved solutions. Originally due to John Holland (1975), the three major processes of the genetic algorithm are *selection*, *crossover* and *mutation*. Holland has shown that genetic algorithms optimize on the tradeoff between searching for new solutions and making use of solutions that have worked well in the past.<sup>5</sup>

The genetic algorithm approach differs from the gradient descent method in that it begins with not one, but a large set of randomly generated candidate solutions known as “bit strings” because they are typically encoded using a binary alphabet.

In our application of the genetic algorithm, each string represents a candidate coefficient vector that gets used in combination with either the polynomial or neural network specification for the conditional expectation on the right hand side of the Euler equation (3). The elements of each string thus consist of a set of six different coefficients  $\{g_i\}_{i=0}^5$ , one for each of the six coefficients in the polynomial or neural network specifications, equation (5) or (6). The elements of our strings are *real-valued*, floating point numbers, not the binary encodings that are traditionally used in genetic algorithm applications.<sup>6</sup> The population size of the parameter vectors is kept constant over many generations (i.e. iterations) of the genetic algorithm. We consider populations of size 40. The initial population of strings (candidate parameter vectors) is randomly generated.

---

<sup>5</sup> Mitchell (1996) provides a good introduction to genetic algorithms.

<sup>6</sup> Real-coded genetic algorithms are increasingly being used in genetic algorithm applications as they avoid the time consuming conversion into and out of binary representations. In many applications, real-coded genetic algorithms have been shown to outperform binary-coded genetic algorithms [ Davis (1991)]. Michalewicz (1994) and Herrera, Lozano and Verdegay (1998) discuss the advantages of real-coded genetic algorithms.

A detailed description of our implementation of the genetic algorithm is provided in the Appendix. Here we describe the genetic algorithm search process we use in more general terms. Starting with some initial, randomly chosen population of strings, the genetic algorithm proceeds in the following sequence of steps.

The first step involves selection based on relative fitness. Two strings are selected at random with replacement from the entire population of strings. The "fitness" value of each string is determined using a given objective function. In our application, the fitness value of a string is inversely proportional to the sum of squared Euler equation residuals which is computed by using the string's approximation model in the Euler equation and simulating the implied system for a sample of  $T=2,000$  iterations. The fitness values of the selected pair of strings are compared. The more fit of the two strings, i.e. the string with the lowest sum of squared errors (or "highest fitness"), is retained for "breeding purposes." This simple tournament selection process is intended to mimic the evolutionary notion of survival of the fittest.<sup>7</sup>

The tournament selection process is repeatedly applied so as to obtain a breeding population that is equal in size to the original population of candidate solutions. A consequence of this tournament selection process is that the fitness levels of the resulting breeding or "parent" population will, on average, be higher than the fitness levels of the preceding generation of strings.

The strings in this breeding population are then randomly paired. The *crossover* operation is then applied to each pair of strings with some probability  $\phi > 0$ . If a pair of strings is selected for crossover, then one of three different crossover operators is employed, with each method having equal probability of being chosen. These various methods of crossover have all been shown to have certain strengths relative to other methods, and are particularly well suited to our real-coded genetic algorithm. A crossover

---

<sup>7</sup> The simple tournament selection process that we use has several advantages over the more traditional roulette-wheel selection method. See, e.g. the discussion in Mitchell (1996).

operation involves swapping the elements of the two candidate vectors in various ways so as to create two new candidate vectors that may yield improved fitness values (more accurate approximations). The essential idea is that elements in certain positions of one string might work better in combination with certain elements in another string; the crossover operation provides a means of implementing such possibilities.

Following crossover, the recombined string pairs as well as the string pairs in the breeding population that were not subject to crossover are then subjected to mutation. The *mutation* operation is applied to each element of each string with some small probability  $\mu > 0$ . We used a *non-uniform* mutation operation that allows for large mutations of coefficient values in the initial stages of the search, but which results in increasingly smaller and localized mutations at later stages of the search, so as to allow for fine tuning of the coefficient estimates. This kind of mutation operation is well suited to the real-coded genetic algorithm that we employ. The string pair that results from application of the crossover and mutation operations can be thought of as the two "offspring" or "children" of the original pair of "parent" strings.

The final step is a selection tournament among the two parents and their two children based on the fitness values of each member in the "family" of four coefficient vectors. The two fittest members of the family survive, and take their place as members of the next generation of candidate strings. The other two strings are discarded. This last step, which Arifovic (1994) has termed the "election operator" is designed to keep the search directed toward solutions with increasingly higher fitness values. Without this operator, children with fitness values lower than their parents would be allowed into the next generation of candidate strings.<sup>8</sup>

---

<sup>8</sup> The election operator also serves to endogenously contain the destructive effects of the mutation operator as one gets closer to an optimal solution.

The above process is repeated until a new generation, equal in size to the previous generation has been constructed. Some members of this new generation are “parents” from the previous generation and others are new "children."

After a given number of generations, the best coefficient set (the one with the best fitness value), is chosen as the coefficient set for optimizing the model. Alternatively, the process can stop when all coefficient strings are identical and have converged on a solution or the fitness values of the strings fail to change by a given amount.<sup>9</sup>

While it is clear from the process described above that only above average coefficient strings will survive successive populations, one can insure that the one "best string" always survives from one generation to the next by applying elitism, which we employ in our algorithm.<sup>10</sup> A given coefficient string, say,  $\tilde{G}$ , from generation  $G$ , ranking first in this generation (in fitness terms), may not be chosen for application of the genetic operations leading to the creation of generation  $G+1$ . Under the version of elitism used here, if the fitness criterion, as applied to all members of generation  $G+1$ , does not yield a coefficient string with fitness higher than  $\tilde{G}$ , then one copy of  $\tilde{G}$  is made and carried over intact into generation  $G+1$ , replacing the least fit coefficient string in  $G+1$ .

As Hassoun (1995) points out, of the three genetic operators, the crossover operator is the most crucial for obtaining global results. It is responsible for mixing the partial information contained in the strings of the population. The mutation operator, by contrast, diversifies the search and introduces new strings into the population, in order to fully explore the search space. The selection operations insure that fitness pressures are kept high in the direction of the search goal.

In our implementation of the genetic algorithm, we set the maximum number of generations for finding the “fittest” string at 150.<sup>11</sup> We will refer to the coefficient string

---

<sup>9</sup> A more formal mathematical representation of the genetic algorithm appears in the Appendix.

<sup>10</sup> See Ruldoph (1994) on the importance of elitist selection for global optimization by genetic algorithms.

<sup>11</sup> Further details concerning our implementation of the genetic algorithm are provided in the appendix.

with the best fitness in generation 150 as the outcome of the 'pure' genetic algorithm optimization.

However, in all of our simulations, we continued the search for the best coefficient vector even further, taking the best-of-generation-150 coefficient vector from the "pure" genetic algorithm search and using this single string to initialize a gradient-descent algorithm, which had the same goal as the genetic algorithm search, to minimize the sum of squared errors of the Euler equation residual over the sample size of 2,000 observations. This gradient-descent algorithm, based on a quasi-Newton method<sup>12</sup>, was allowed to iterate for up to 3,000 iterations so as to fine-tune the best string resulting from the genetic algorithm search. The gradient-descent algorithm was allowed to terminate prior to 3,000 iterations if the minimal step change ( $1 \cdot 10^{-8}$ ) in the line search for the coefficient vector led to no change in a finite difference gradient calculation *and* the objective function (the sum of squared errors of the Euler equation residual) had achieved a precision of at least  $1 \cdot 10^{-4}$  (four digit accuracy).

We refer to this procedure as a "hybrid" genetic algorithm/gradient-descent algorithm. The hybrid algorithm thus consists of a preliminary population-based "global" search process, the result of which is used to initialize a second-stage "local" gradient-descent algorithm.

We now examine the outcome of using our hybrid genetic algorithm/gradient-descent algorithm in combination with the two different methods of parameterizing expectations -- polynomial and neural network specifications -- for given parameter values of the stochastic model:

$$a = .33, d = \{1, 0\}, r = .95.$$

---

<sup>12</sup> The quasi-Newton method we used makes use of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula for updating the approximation of the Hessian matrix. This approach is described, e.g. in Judd (1998).

For approximating the exact solution, where  $\delta=\tau=1$ , we use two values of  $b \in \{.95, .98\}$ , and three values of  $s \in \{.01, .05, .10\}$ . For cases where  $d = 0$  and  $t \in \{0.5, 1.5, 3.0\}$ , we again use two values of  $b \in \{.95, .98\}$  but only two values of  $s \in \{.02, .10\}$ . These parameter value choices allow us to make direct comparisons with the results reported in Taylor and Uhlig (1990).

### 3. Simulation Results

Tables 1 and 2 report simulation results from the genetic algorithm/gradient-descent search algorithm for both the polynomial and neural network parameterization methods in the case where  $\tau=\delta=1$ , and an exact analytic solution exists. This special case provides us with a useful benchmark in which to assess the accuracy of our two approximation methods, and we do so using a dimension-free measure, as advocated by Judd (1992) -- the  $\log_{10}$  average relative squared error.

Tables 3 through 7 provide simulation results from the genetic algorithm/gradient-descent search algorithm for the polynomial and neural network approximations using the Taylor-Uhlig (1990) version of the model where there is no depreciation,  $\delta=0$ , and where  $\tau=0.5, 1.5, \text{ or } 3.0$ , so that no closed-form solution exists.<sup>13</sup> In these cases, we compare our approximated solutions with the methods used by other researchers, as reported in Taylor and Uhlig (1990) using the same set of summary statistics to assess solution accuracy.

#### 3.1 Summary Statistics

For both the polynomial and neural network approximation methods, and for all versions of the model, we compute the mean sum of squared parameterized expectation (PE) errors for each solution that results from our combined genetic algorithm/gradient-descent search process. We define this statistic as the Marcet PE-Error:

---

<sup>13</sup> Matlab codes for our approximation method and simulation exercises may be found on McNelis's website, [www.georgetown.edu/mcnelis](http://www.georgetown.edu/mcnelis), under Recent Research, and the title of this paper.

$$\text{PE-Error} = \sum u_t^2 / T = \sum [l_{t+1} - \hat{y}(\hat{g}; k_{t-1}, q_t)]^2 / T \quad (7)$$

where  $l_{t+1} = [c_{t+1}^{-\tau} (q_{t+1} k_t^a + 1 - d)]$ , and  $T$  is the sample size, set at 2000.

In addition we present the volatility of the consumption series (denoted as "convol" in the tables), which is simply the standard deviation of the Hodrick-Prescott (HP) filtered series. We also report the ratio of the variance of investment to the variance of the first difference of consumption, denoted as the "i-c ratio" in the tables.

For the case where  $\tau = d = 1$ , we are able to compute an exact solution. We thus compute the correlation coefficient of the consumption series obtained under the approximation method with the consumption series obtained using the exact solution. This statistic is labeled "corr with exact" in Table 1.

In this case where the exact solution is known, we also compute an approximation error statistic that is based on the difference between the actual and approximated policy functions for consumption. The statistic we use is the  $\log_{10}$  average relative squared error, given by:

$$e(h) = \log_{10} \frac{1}{N_k N_q} \sum_k \sum_q \left( \frac{\hat{h}(k, q) - h(k, q)}{h(k, q)} \right)^2 \quad (8)$$

where  $h(k, q)$  is the true policy function, in this case,  $h(k, q) = (1 - ab)qk^a$ , and  $\hat{h}(k, q)$  is the policy function recovered from either the polynomial or neural network approximation method for parameterized expectations.<sup>14</sup> The value of this accuracy statistic,  $e(h)$ , is obtained by evaluating both policy functions over a grid of  $N_k$  and  $N_\theta$  values for  $k$  and  $\theta$ .

In particular, we generated values for  $e_t$  for 80 equally spaced points between the interval  $[-2S, +2S]$ . These grid points were then converted into grid points for  $\ln q$  by the long-run relation,  $\ln q = \frac{1}{1 - \tau} e$ , and for  $q$  by simply taking the exponent of  $\ln q$ . To generate grid points for  $k$ , we made use of the grid points for  $q$  and the long run relation

---

<sup>14</sup> This error metric is similar to the one advocated by Judd (1992) to assess approximation accuracy.

between  $k$  and  $q$ ,  $k = (abq)^{1/(1-a)}$ . With 80 grid points for  $k$  and  $q$ , the error metric,  $e(h)$ , is thus evaluated over 6,400 different combinations of  $k$  and  $q$ . We note that for each value of  $S$ , the implied grid of  $k$  values lies within the set of feasible values for  $k$ :  $[0, (q/d)^{1/(1-a)}]$ .

The  $\log_{10}$  average relative squared error,  $e(h)$ , was chosen because it provides an easily interpretable measure of accuracy, expressing the approximation error as a fraction of consumption. A  $\log_{10}$  squared error of -2 represents an accuracy rate of 1 in 100, implying that the approximation error costs \$1 for every \$100 in consumption expenditures; a  $\log_{10}$  squared error of -3 represents an accuracy rate of 1 in 1,000.

For the other case where  $t \neq 1, d = 0$ , and no closed form solution exists, we use the following four summary statistics to evaluate the accuracy of our results: (1) the Den Haan-Marcet statistic, (2) the  $TR^2$  statistic, (3) the  $R^2$  statistic and (4) the ratio of the variance of investment to the variance of the change in consumption. All of these statistics are described and were reported in Taylor and Uhlig (1990) for a variety of different solution methods. Therefore, we shall only briefly review these statistics here.

The Den Haan-Marcet (1994) accuracy statistic, "DM-stat," in the tables, is computed in the following way:

$$\begin{aligned} h_t &= bc_t^{-t} (aq_t k_{t-1}^{a-1} + 1 - d) - c_{t-1}^{-t} \\ \hat{a} &= (x'x)^{-1} x'h \\ \text{DM-stat} &= \hat{a}'(x'x)(x'xh^2)^{-1}(x'x)\hat{a} \end{aligned} \tag{9}$$

Here,  $x$  is a matrix of instrumental variables, which in our case consists of a constant and lagged values of consumption and the productivity shock. The "DM-stat" statistic is distributed as a Chi-square variable with degrees of freedom equal to number of

instruments used, under the null-hypothesis of an accurate approximation to the optimal path.

In calculating this statistic for our various solutions, we used as instruments, five lags of consumption,  $c$ , five lags of the productivity shock,  $q$ , and a constant term for each sample size of 2,000 observations. The DM-statistics reported in Taylor and Uhlig (1990) were calculated using this same set of 11 instruments, thus facilitating a comparison of our results with the results of the various other solution methods reported in Taylor and Uhlig (1990). Under the null hypothesis of an accurate approximation, the DM-stat has an asymptotic  $\chi^2(11)$  distribution with critical values [3.81, 21.92] at the five percent level, and critical values [3.05, 24.72] at the one percent level of significance.

The  $TR^2$  statistic ("tr2stat" in the tables) is computed from a regression of the productivity shock,  $\varepsilon$ , on five lags of consumption, capital, and  $q$  (15 lags total), again as in Taylor and Uhlig (1990). This test statistic is used to assess the martingale difference property of the *productivity* shocks,  $E_{t-1} \varepsilon_t = 0$ , and thus serves as another measure of the accuracy of the approximated solution. The following system describes the calculation of the  $TR^2$  statistic

$$\begin{aligned} \hat{\varepsilon}_t &= x_t \hat{b}, \\ \hat{b} &= (x_t' x_t)^{-1} x_t' \varepsilon_t, \\ \text{tr2stat} &= T \frac{[\sum (\varepsilon_t - \bar{\varepsilon}_t)(\hat{\varepsilon}_t - \bar{\hat{\varepsilon}}_t)]^2}{\sum (\varepsilon_t - \bar{\varepsilon}_t)^2 \sum (\hat{\varepsilon}_t - \bar{\hat{\varepsilon}}_t)^2}, \end{aligned} \tag{10}$$

Here again,  $T$  denotes the number of observations in the regression sample, taken to be 2,000, and  $x_t$  represents the 15x1 vector of lagged values for consumption, capital and  $q$ . Under the null hypothesis that the productivity shock possesses the martingale property,

this test statistic has an asymptotic  $\chi^2(15)$  distribution. The critical bounds at the 5 percent significance level are [6.26, 27.49].

The  $R^2$  statistic ("rsqstat" in the tables) comes from a regression of the first difference of consumption on lagged consumption and capital, again using the sample of 2,000 observations. This test statistic provides a simple test of the random walk hypothesis for consumption in the simulated data. An  $R^2$  close to zero is taken as support for the random walk hypothesis.<sup>15</sup>

### 3.2 The case where $t=d=1$ .

Table 1 presents the various test statistics for the case of the known closed form solution where  $t = d = 1$ ,  $b \in \{.95, .98\}$ , and  $s \in \{.01, .05, .10\}$ . Panel A of Table 1 reports benchmark values for all summary statistics using the known, exact solution. Panels B and C do the same for the neural network and polynomial approximations, respectively.

Consider first, the value of the  $\log_{10}$  average relative squared error,  $e(h)$ , for the two different approximation methods, which provides a metric for assessing the accuracy of the policy functions in this special case where the policy function is known. Panels B and C of Table 1 reveal that the accuracy of both approximation methods is best when  $s = .01$  and that accuracy decreases as  $s$  increases. Indeed, when  $s = .10$  the approximation errors of both methods are rather high.

Notice that the neural network approximation provides the more accurate policy function when  $\sigma = .01$ . while when  $\sigma = .05$ , the neural network and polynomial

---

<sup>15</sup> Since there is zero depreciation, the capital stock would follow a random walk in a linearized expansion around the steady state. Since consumption is a linear function of the capital stock, it too would follow a random walk. Thus testing for a random walk for consumption is not an unreasonable accuracy check under zero depreciation.

approximations achieve roughly the same approximation accuracy. The polynomial approximation yields the more accurate policy function for the case where  $\sigma=.10$ .

Figure 1 depicts the values of  $e(h)$ , for both the neural network (n) and polynomial (p) approximations, for values of  $S \in [.01,.1]$ . For each approximation method, values of  $e(h)$  are presented for the case where  $b = .95$  and  $b = .98$ . For values of  $S < .04$ , the network approximation outperforms the polynomial method, but for relatively high values,  $S > .06$ , the polynomial method outperforms the network model.

This result should not be surprising, since the exact solution itself is a log-linear function. The second-order polynomial approximation is thus an over-parameterization encompassing the true solution, whereas the network is an approximation with an alternative functional form. When the size of the shocks are relatively small, the network serves as a very accurate approximation. As the shocks rise in value and  $k$ , the capital stock, wanders far away from its deterministic steady state value, the polynomial method, which encompasses the true values of the state variables, dominates.

As noted above, the parameterized expectation error criterion is based on a simulation over a sample size of 2000. The  $\log_{10}$  average relative squared error statistic,  $e(h)$ , by contrast, is based on a one-step simulation with alternative grids or starting conditions for  $k$  and  $\theta$ . Since the parameterized expectation approach is designed to minimize the parameterized expectation error, the PE-Error statistic, reported in Table 1, is the objective which is minimized by the genetic algorithm/gradient-descent method. We find similar values for this statistic using either neural network or polynomial approximations.

Under both methods, the approximated paths overestimate the investment/consumption volatility ratio (i-c ratio) as well as the volatility of consumption itself (con vol). Nevertheless, there is a very high correlation between the approximated consumption path and the "true" path as indicated by the correlation coefficient reported as "corr w exact."

Table 2 reports the six coefficient estimates for the approximation function under the two specifications, polynomial or neural network expansion, for all of the cases where  $t = d = 1$ . The reported coefficients are based on the best expectation error minimization chosen from 10 different runs of the combined genetic algorithm and gradient-descent estimation procedure. Each run was begun with a different set of randomly initialized coefficient vectors. Below the best coefficient values in Table 2 are standard deviations (*in italics*) which reveal the extent to which the best coefficient values differ from the coefficient values obtained from all 10 runs. There are two sets of standard deviations for each set of coefficient estimates. The first set of standard deviations is based solely on the final coefficient estimates obtained from ten runs of the genetic algorithm alone (the 'pure' genetic algorithm). Each run of the genetic algorithm consists of 150 generations. The second set of standard deviations is based on the ten sets of coefficients obtained at the completion of the hybrid genetic algorithm/gradient-descent estimation procedure.

Notice that many of the coefficient estimates for both the neural network and polynomial specifications lie within two standard deviations of the best coefficient estimates from all 10 runs. This finding indicates that the combination of a "global" genetic search with "local" gradient-descent methods, for both the network and polynomial approximations, yields coefficient estimates within a relatively close range of the "best" set of estimated coefficient values.

Notice too, that the coefficient estimates for both the neural network and polynomial specifications do change, sometimes dramatically, with changes in the parameter values of the stochastic growth model,  $b, s, t$ . While such changes are not unexpected given the nonlinearity of the model, they may also arise from our use of an overparameterized specification for the expectations function in this version of the model where the exact (first-order) solution is known.

The main findings of Tables 1 and 2 are that the neural network specification often provides at least as accurate an approximation as the second-order polynomial

specification, even when the exact solution is itself a simple first-order polynomial. Furthermore, multiple, randomly initialized runs of our combination genetic algorithm/gradient descent search procedure often results in similar coefficient estimates for the parameterized expectations function.

### *3.3 The Taylor-Uhlig (1990) Model*

Table 3 presents the accuracy and diagnostic test statistics for the two approximation methods (polynomial and neural network) as applied to the Taylor-Uhlig (1990) stochastic growth model, under alternative assumptions for  $t$  and  $s$ . Since there is no depreciation ( $d = 0$ ) and  $t \neq 1$  in the Taylor-Uhlig version of the model, the dynamic process in this model differs somewhat from the model where  $d = t = 1$  and an exact closed form solution could be obtained.

Table 3 shows that the Marcet PE-Errors are quite small for both models, usually less than .0001, under different parameter configurations. These PE-errors should be small as they are the statistics the genetic algorithm/gradient-descent search procedure is seeking to minimize.

The "accuracy" statistic we use in this version of the model is the DM -statistic. In Table 3, DM-statistics that violate the critical bounds at the 1 percent significance level are italicized. We see that the neural network specification violates the accuracy criterion at the one percent level in three out of twelve cases, and never for versions of the model where  $b = .98$ , whereas the polynomial specification violates this criterion in five cases out of twelve.

The “rsqstat” values are quite low for both specifications, with the exception of  $t = 3$  and  $s = .1$ , for the polynomial specification under  $b = .95$  and  $.98$ . These results are thus consistent with “random walk” consumption behavior, under zero depreciation of the capital stock.

The “tr2stat” values are always within the Chi-square accuracy bounds for both the polynomial and neural network approximation methods. However there are sharp divergences across the various cases in the investment/consumption volatility ratios as well as in the direct measures of consumption volatility itself.

Tables 4 and 5 present the “best” coefficient estimates (those which minimized the expectations error) over 10 runs of our search algorithm for the two methods of approximating parameterized expectations. As in Table 2, Tables 4 and 5 present two sets of standard deviations (in italics) which reveal the extent to which these “best” coefficient estimates differ from the estimates obtained from all 10 runs of the pure genetic algorithm (first set of standard deviations), as well as from the coefficients obtained at the end of the combined genetic algorithm/gradient descent search procedure (second set of standard deviations). Both the neural network (Table 4) and polynomial (Table 5) approximations show some degree of consistency, in the sense that our search algorithm delivers coefficient estimates in a relatively close range of the very best estimated coefficient values, as revealed by the standard deviations reported in these two tables.

Tables 6 and 7 summarize the accuracy and diagnostic test statistics from our neural network and polynomial approximations (Duffy/McNelis-NN and Duffy/McNelis-PA), and compare these statistics with those obtained from a subset of the other solution methods presented in Taylor and Uhlig (1990). In particular, we compare our solution

method with the log-Linear Quadratic (log-LQ) and linear quadratic LQ solution methods of Christiano and McGrattan, the backsolving methods of Ingram and Sims, the parameterized expectations approach of Den Haan and Marcet, and the quadrature method of Tauchen, all of which are discussed in the January 1990 edition of the *Journal of Business and Economic Statistics*. Table 6 reports statistics for various versions of the model when  $\sigma=.02$ , and Table 7 does the same for the case when  $\sigma=.10$ . Inspection of these two tables indicate that the parameterized expectation approach using neural network and polynomial approximations and our hybrid estimation method, compare favorably on many dimensions with these alternative and more commonly used methods.

In those cases where the DM-statistics for our solution methods violate the accuracy criterion, these statistics are not as high, for example, as those obtained by Tauchen (1990) using a quadrature method. Even the results reported by Den Haan and Marcet (1990) violate the DM-statistic critical bounds in three out of the ten parameter configurations.<sup>16</sup>

Similarly, for our solution method, the investment-consumption volatility ratio is only an "outlier" in one case, for the neural network specification, with  $t = .5$  and  $S = .1$ . The  $TR^2$  and  $R^2$  statistics from our solution method also compare favorably with the values obtained using these other solution methods.

The average time for the genetic algorithm search and gradient-descent optimization for one parameter configuration was 13.55 hours, on a SUN Enterprise 4000

---

<sup>16</sup> The Den Haan and Marcet accuracy statistics reported in Den Haan and Marcet (1994) are much better than those reported Den Haan and Marcet (1990). However, Den Haan and Marcet consider different parameterizations of the stochastic growth model in these two studies. In particular, in Den Haan and Marcet (1994), depreciation is not set equal to zero as it is in the Taylor-Uhlig (1990) version of the model considered by Den Haan and Marcet (1990).

with four 250 megahertz processors and 250 megabytes of RAM, with version 5.2 of Matlab. The time was generally the same for both the network and polynomial approximation methods.

While this runtime may seem exorbitantly long in light of the computing times reported in seconds in Taylor and Uhlig (1990), Table 15, our run times are not directly comparable. The genetic algorithm evaluates each case or parameter configuration, with a sample size of 2000, 40 different times in each generation, for 150 generations. An additional 40 evaluations of the case take place within each generation during the repeated processes of cross-over and mutation. It is only after this process is completed that the gradient-descent optimization process begins and continues until convergence is achieved. The experiment is then repeated an additional ten times to find the best coefficient values for the parameterized expectation function.

The global search/local gradient-descent optimization process is clearly not going to be the fastest solution method. It is a lengthy search process requiring multiple evaluations, but it does offer both accuracy and the added confidence that comes from conducting a global search.

#### **4. Conclusion**

This paper has shown that a neural network approach to the parameterized expectation solution method may be a useful alternative to polynomial expansions, and that a sensible way to proceed is to estimate the parameters of a parameterized expectations model using a combined genetic algorithm/gradient-descent method. For the special case of the one-sector stochastic growth model where an exact solution is known, we demonstrate that our modified parameterized expectation solution methods are highly

accurate. Many of the solutions we obtain using our approach for several other versions of the model in which there is no depreciation and the coefficient of relative risk aversion is set to 0.5, 1.5, or 3.0, pass an accuracy test proposed by Den Haan and Marcet (1994) and appear to be quite reasonable in comparison with other solution methods. We conclude that our hybrid genetic algorithm/gradient-descent parameterized expectations method should be used at least as a check on other solution methods.

In more complicated models, with even higher degrees of nonlinearity due to more complex utility functions, or in multi-sector models with higher dimensions, the parameterized expectations algorithm based on pure gradient-descent (or non-linear least squares) methods may fail to converge. For this reason, many researchers have turned to Tauchen-Hussey (1991) quadrature-based methods which rely on discretizing the state-space. Our method may extend the applicability of the parameterized expectations algorithm to these more complex higher dimensional cases. We leave such an analysis to future research.

### **Acknowledgements**

We thank five anonymous referees and participants at the 1997 Society for Economic Dynamics and 1998 Society for Computational Economics conferences for helpful comments and suggestions on earlier drafts.

## Appendix: Detailed Discussion of our Implementation of the Genetic Algorithm

The genetic algorithm starts with the following objective function and constraint:

$$\text{Min} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\hat{y} = g(x | \hat{g})$$

Here,  $y_i$ , denotes the actual value of the right hand side of the Euler equation (2) for observation  $x_i = \{k_{i-1}, q_i\}$  and the estimated vector based on the approximation is denoted by  $\hat{y}_i$ . We chose to consider  $n=2,000$  for each candidate solution. The genetic algorithm consists of the following eight steps.<sup>17</sup>

Step one: Create an initial population  $p$  of coefficient vectors at generation 1:

$$\begin{bmatrix} \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_k \end{bmatrix}_1 \begin{bmatrix} \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_k \end{bmatrix}_2 \begin{bmatrix} \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_k \end{bmatrix}_i \dots \begin{bmatrix} \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_k \end{bmatrix}_p$$

where  $p$  is an even number (in our case  $p=40$ ) and  $k$  is the number of coefficients. (in our case  $k=6$ ). The initial population of size  $p \times k$  is created using random draws from a standard normal distribution. Alternatively, one may impose restrictions on the sign or size of the parameters, fixing some of them and setting others randomly within a certain range.

Step two: Select two pairs of coefficient vectors from the population at random with replacement. Evaluate the fitness of these four coefficient vectors according to the objective function given above. Vectors that come closer to minimizing the sum of squared errors receive higher fitness values. Next, conduct a simple fitness tournament between the two pairs of vectors: the winner of each tournament is the vector with the best fitness (lowest sum of squared errors). These two winners,  $(i,j)$ , are retained for "breeding purposes" These two winning coefficient vectors are depicted below:

<sup>17</sup> This algorithm is similar to the one used by Bullard and Duffy (1999).

$$\begin{bmatrix} \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_k \end{bmatrix}_i \begin{bmatrix} \hat{g}_1 \\ \hat{g}_2 \\ \hat{g}_3 \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_k \end{bmatrix}_j$$

Coefficient vectors  $i$  and  $j$  are referred to as the "parent" vectors,  $P(i)$  and  $P(j)$ .

Step three: Allow crossover to be performed on each pair of parent coefficient vectors,  $i$  and  $j$ , with some fixed probability  $\phi > 0$  (we set  $\phi = .95$ ). If crossover is to be performed, we use one of *three* different crossover operations, with each method having an equal (1/3) probability of being chosen. These three crossover operators are:

1. *Shuffle crossover*. For each pair of vectors,  $k$  random draws are made from a binomial distribution. If the  $k^{\text{th}}$  draw is a 1, coefficients  $\hat{g}_{i,k}, \hat{g}_{j,k}$  are swapped; otherwise no change is made.
2. *Arithmetic crossover*. For each pair of vectors, a random number is chosen,  $\lambda \in (0,1)$ . This number is used to create two new parameter vectors that are linear combinations of the two parent vectors, with elements:  $\lambda \hat{g}_{i,k} + (1-\lambda) \hat{g}_{j,k}, (1-\lambda) \hat{g}_{i,k} + \lambda \hat{g}_{j,k}$ .
3. *Single-point crossover*. For each pair of vectors, an integer  $I$  is randomly chosen from the set  $[1, k-1]$ . The two vectors are then cut at integer  $I$  and the coefficients to the right of this cut point,  $\hat{g}_{i,I+1}, \hat{g}_{j,I+1}$ , are swapped.

In binary-coded genetic algorithms, single-point crossover is the standard crossover method. There is no consensus in the genetic algorithm literature on which crossover method is best for *real-valued* encodings. Our decision to adopt these three different approaches follows from the analysis and recommendations of Michalewicz (1996) and Herrera, Lozano and Verdegay (1998).

Following application of the crossover operation, each pair of parent vectors is associated with two children coefficient vectors, which we denote by  $C1(i)$  and  $C2(j)$ . If crossover has been applied to the pair of parents, the children vectors will generally differ from the parent vectors.

Step four: With some small and probability,  $m > 0$ , that decreases over time, each element (coefficient) of the two "children vectors" is subjected to a mutation. The probability that

each element is subjected to mutation in generation  $G=1,2,\dots,150$  was given by  $m_g = .15 + .33/G$ .

If mutation is to be performed on a vector element (coefficient), we used the following non-uniform mutation operation, due to Michalewicz (1996). Begin by randomly drawing two real numbers,  $r_1$  and  $r_2$  from the  $[0,1]$  interval and one random number,  $s$ , from a standard normal distribution. The mutated value of the coefficient  $\check{g}$ , is then given by:

$$\check{g} = \begin{cases} \hat{g} + s[1 - r_2^{(1-t/T)^b}] & \text{if } r_1 > .5 \\ \hat{g} - s[1 - r_2^{(1-t/T)^b}] & \text{if } r_1 < .5 \end{cases}$$

where  $t$  is the generation number,  $T$  is the maximum number of generations, and  $b$  is a parameter that governs the degree to which the mutation operator is non-uniform. We set  $b=2$ , and  $T=150$ . Notice that the probability of creating a new coefficient via mutation that is far from the current coefficient value diminishes as  $t \rightarrow T$ . This mutation operation is non-uniform since, over time, the algorithm is sampling increasingly more intensively in a neighborhood of the existing coefficient values. This more localized search allows for some fine tuning of the coefficient vector in the latter stages of the search, when the vectors should be approaching an optimum.

Step five: Election tournaments. Following the mutation operation, the four members of the “family” (P1, P2, C1, and C2) engage in a fitness tournament. The children are evaluated by the same fitness criterion used to evaluate the parents. The two vectors with the best fitness (lowest sum of squared errors) whether parents or children, survive and pass to the next generation, while the two with the worst fitness are extinguished.

Step six: Repeat the above process, with parent vectors  $i$  and  $j$  returning to the population pool for possible selection again, until the next generation is populated with  $p$  coefficient vectors.

Step seven: Elitism. Once the next generation is populated, introduce elitism. Evaluate all the members of the new generation and the current generation according to the fitness criterion. If the best member of the “older generation” dominates the best member of the “new generation,” then the member of the older generation displaces the worst member of the new generation, and is thus eligible for selection in the coming generation.

Step eight: Continue the process for a maximum of  $T$  generations. We set  $T=150$ . Evaluate convergence by the behavior of the best member of each generation, according to the fitness criterion.

Following  $T=150$  generations, we used the best-of-generation-150 vector (the vector with the highest fitness value) to initialize a quasi-Newton, gradient-descent algorithm, which we allowed to run for up to 3,000 iterations.

## References

Arifovic, J., 1994, Genetic algorithm learning and the cobweb model, *Journal of Economic Dynamics and Control* 18, 3-28.

Barron, A.R., 1993, Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory* 39, 930-945.

Beaumont, P.M. and P.T. Bradshaw, 1995, A distributed parallel genetic algorithm for solving optimal growth models, *Computational Economics* 8, 159-179.

Beaumont, P.M. and P.T. Bradshaw, 1996, A distributed parallel genetic algorithm: An application from economic dynamics, in: M. Gilli, ed., *Computational Economic Systems: Models, Methods and Econometrics*, (Kluwer, Boston).

Bishop, C. M., 1995, *Neural networks for pattern recognition* (Oxford University Press, New York).

Bullard, J. and J. Duffy, 1999, Learning and excess volatility, working paper, University of Pittsburgh.

Chyi, Y.L., 1997, Solving the stochastic growth model by a neural network learning approach, working paper, National Tsing Hua University.

Christiano, L.J. and J.D.M. Fisher, 1997, Algorithms for solving dynamic models with occasionally binding constraints, NBER technical working paper no. 218.

Davis, L., 1991, ed., *Handbook of genetic algorithms*, (Van Nostrand Reinhold, New York).

Den Haan, W.J. and A. Marcet, 1990, Solving the stochastic growth model by parameterizing expectations, *Journal of Business and Economic Statistics* 8, 31-34.

Den Haan, W.J. and A. Marcet, 1994, Accuracy in simulations, *Review of Economic Studies* 61, 3-17.

Dorsey, R E. and W. J. Mayer, 1995, Genetic algorithms for estimation problems with multiple optima, nondifferentiability, and other irregular features, *Journal of Business and Economic Statistics* 13, 53-66.

Hassoun, M. H., 1995, *Fundamentals of artificial neural networks*, (MIT Press, Cambridge, MA).

Herrera, F., M. Lozano and J.L. Verdegay, 1998, Tackling real-coded genetic algorithms: Operators and tools for behavioral analysis, *Artificial Intelligence Review* 12, 265-319.

- Holland, J., 1975, *Adaptation in natural and artificial systems*, (University of Michigan Press, Ann Arbor), Reprinted in 1992, (MIT Press, Cambridge, MA).
- Hornik K., M. Stinchcombe, and H. White, 1989, Multi-layer feedforward networks are universal approximators, *Neural Networks* 2, 359-366.
- Judd, K.L., 1992, Projection methods for solving aggregate growth models, *Journal of Economic Theory* 58, 410-452.
- Judd, K.L., 1998, *Numerical Methods in Economics*, (MIT Press, Cambridge, MA).
- Marcet, A., 1988, Solving nonlinear models by parameterizing expectations, working paper, Carnegie-Mellon University.
- Michalewicz, Z., 1996, *Genetic algorithms + data structures = evolution programs*, third edition, (Springer-Verlag, New York).
- Mitchell, M., 1996, *An introduction to genetic algorithms*, (MIT Press, Cambridge, MA).
- Rudolph, G., 1994, Convergence analysis of canonical genetic algorithms, *IEEE Transactions on Neural Networks* 5, 96-101.
- Sargent, T.J., 1987, *Dynamic macroeconomic theory*, (Harvard University Press, Cambridge, MA).
- Sargent, T. J., 1993, *Bounded rationality in macroeconomics*, (Oxford University Press, New York).
- Schmertmann, C.P., 1996, Functional search in economics using genetic programming, *Computational Economics* 9, 275-298.
- Tauchen, G., 1990, Solving the stochastic growth model by using quadrature methods and value-function iterations, *Journal of Business and Economic Statistics*, 8, 49-51.
- Tauchen, G. and R.M. Hussey, 1991, Quadrature-based methods for obtaining approximate solutions to non-linear asset-pricing models, *Econometrica* 59, 371-396.
- Taylor, J.B. and H. Uhlig, 1990, Solving nonlinear stochastic growth models: A comparison of alternative solution methods, *Journal of Business and Economic Statistics* 8, 1-17.

**Table 1: Accuracy/Diagnostic Statistics for the Model with tau=delta=1**

**A: Benchmark Values for Exact Solution**

	<u>Beta = .95</u>				<u>Beta = .98</u>		
	<u>Sigma:</u>				<u>Sigma:</u>		
	0.01	0.05	0.1		0.01	0.05	0.1
i-c ratio	0.733337	0.733837	0.731722	i-c ratio	0.799104	0.803937	0.801619
con vol	0.006551	0.030116	0.065364	con vol	0.005721	0.03014	0.065416

**B: Solutions for Network Approximation**

	<u>Beta = .95</u>				<u>Beta = .98</u>		
	<u>Sigma</u>				<u>Sigma</u>		
	0.01	0.05	0.1		0.01	0.05	0.1
e(h)	-1.36	-0.2626	1.07	e(h)	-1.52	-0.236	1.12
PE-Error	1.68E-05	5.48E-05	0.001207	PE-Error	3.82E-07	4.85E-05	0.000986
i-c ratio	1.027208	1.240308	1.285603	i-c ratio	0.982809	0.984299	1.064046
con vol	0.005929	0.027166	0.058701	con vol	0.00549	0.028952	0.062435
corr w exact	0.99933	0.999856	0.999002	corr w exact	0.999988	0.999894	0.998919

**C: Solutions for Polynomial Approximation**

	<u>Beta = .95</u>				<u>Beta = .98</u>		
	<u>Sigma</u>				<u>Sigma</u>		
	0.01	0.05	0.1		0.01	0.05	0.1
e(h)	-0.4419	-0.3842	0.219	e(h)	-0.4814	-0.3943	0.2327
PE-Error	1.75E-06	7.12E-05	0.000337	PE-Error	0.000126	9.09E-06	0.000338
i-c ratio	1.155243	1.248969	1.255044	i-c ratio	0.940838	0.987292	1.009726
con vol	0.005167	0.027151	0.058835	con vol	0.029052	0.028934	0.062663
corr w exact	0.999925	0.999825	0.999626	corr w exact	0.999725	0.999979	0.999618

**Table 2: Coefficients of Parameterized Expectations Models (where tau=delta=1)**

**A. Neural Network Coefficients and Standard Deviations**

*(Standard Deviations In Italics, first row based on ga estimates only)*

beta	sigma	constant	Neuron 1		Neuron 2		
			coefficient	k(-1)	theta	k(-1)	theta
0.95	0.01	16.47942	0.314399	-2.125809	-1.3295	-6.754121	-11.62937
		<i>0.546141</i>	<i>1.011653</i>	<i>9.836234</i>	<i>1.36487</i>	<i>1.760005</i>	<i>0.959442</i>
		<i>61.44795</i>	<i>13.31865</i>	<i>8.510216</i>	<i>1.56951</i>	<i>2579.176</i>	<i>33.28563</i>
0.95	0.05	21.41425	1.206309	-2.826033	-2.10821	-5.417566	-0.605893
		<i>2.002061</i>	<i>0.4385</i>	<i>1.872858</i>	<i>0.43036</i>	<i>4.886588</i>	<i>4.245783</i>
		<i>0.996521</i>	<i>0.514417</i>	<i>0.117762</i>	<i>0.06025</i>	<i>377.8443</i>	<i>8.81035</i>
0.95	0.1	21.02004	1.070417	-3.412623	-1.89004	1.492699	-1.71325
		<i>6.367557</i>	<i>0.619806</i>	<i>8.899572</i>	<i>0.47685</i>	<i>2.423918</i>	<i>4.68467</i>
		<i>9.698343</i>	<i>0.64191</i>	<i>8.111838</i>	<i>0.7752</i>	<i>5.076085</i>	<i>5.384808</i>
0.98	0.01	17.97724	0.959983	-2.833295	-1.72594	-9.910302	-2.879743
		<i>1.096616</i>	<i>0.234255</i>	<i>2.51229</i>	<i>1.05172</i>	<i>1.956257</i>	<i>0.848457</i>
		<i>44.5208</i>	<i>0.47445</i>	<i>3.846435</i>	<i>1.01295</i>	<i>36.68764</i>	<i>84.80875</i>
0.98	0.05	20.8292	0.156844	-3.180089	-1.9474	37.11811	45.87679
		<i>3.389278</i>	<i>1.106713</i>	<i>4.127126</i>	<i>0.82269</i>	<i>2.123386</i>	<i>4.905593</i>
		<i>8.288293</i>	<i>1.004706</i>	<i>1.392792</i>	<i>0.94018</i>	<i>18.65472</i>	<i>20.91533</i>
0.98	0.1	20.10909	1.097249	-2.291695	-2.10609	-17.07265	1.103343
		<i>8.543583</i>	<i>0.981625</i>	<i>9.966193</i>	<i>1.25356</i>	<i>4.406638</i>	<i>1.323581</i>
		<i>10.92276</i>	<i>0.963152</i>	<i>9.64929</i>	<i>1.5527</i>	<i>7.463489</i>	<i>1.395279</i>

**B. Polynomial Coefficients and Standard Deviations**

*(Standard Deviations In Italics, first row based on ga estimates only)*

beta	sigma	constant	First-Degree		Second-Degree		Cross-term ln[k(-1)]ln(theta)
			ln[k(-1)]	ln(theta)	ln[k(-1)]	ln(theta)	
0.95	0.01	2.309697	0.371658	-1.024036	0.398282	1.527809	1.102599
		<i>0.449369</i>	<i>0.642534</i>	<i>1.049247</i>	<i>0.23754</i>	<i>0.247976</i>	<i>0.672973</i>
		<i>0.475185</i>	<i>0.593813</i>	<i>1.052024</i>	<i>0.18854</i>	<i>0.274704</i>	<i>0.668843</i>
0.95	0.05	0.920698	-1.378149	0.16365	-0.15213	0.768553	1.848136
		<i>0.233104</i>	<i>0.367547</i>	<i>0.448049</i>	<i>0.14038</i>	<i>0.129095</i>	<i>0.290893</i>
		<i>0.09266</i>	<i>0.118462</i>	<i>0.341076</i>	<i>0.0379</i>	<i>0.11136</i>	<i>0.220378</i>
0.95	0.1	1.503821	-0.638071	-1.314541	0.084317	1.197503	0.920243
		<i>0.462893</i>	<i>0.623441</i>	<i>0.247738</i>	<i>0.20829</i>	<i>0.111709</i>	<i>0.150018</i>
		<i>0.085839</i>	<i>0.112673</i>	<i>0.113394</i>	<i>0.03662</i>	<i>0.042927</i>	<i>0.073704</i>
0.98	0.01	1.077391	-1.070474	-1.830974	-0.0699	1.516881	0.498521
		<i>0.31096</i>	<i>0.37079</i>	<i>0.96403</i>	<i>0.11985</i>	<i>0.451634</i>	<i>0.613267</i>
		<i>0.256371</i>	<i>0.323214</i>	<i>0.91837</i>	<i>0.10193</i>	<i>0.456472</i>	<i>0.569926</i>
0.98	0.05	1.720697	-0.241182	-1.397958	0.196083	1.310748	0.764183
		<i>0.533738</i>	<i>0.615788</i>	<i>0.715167</i>	<i>0.18087</i>	<i>0.296382</i>	<i>0.454537</i>
		<i>0.354023</i>	<i>0.445132</i>	<i>0.4544</i>	<i>0.1396</i>	<i>0.211436</i>	<i>0.282935</i>
0.98	0.1	1.455901	-0.574595	-1.272582	0.093091	1.176286	0.865596
		<i>0.519941</i>	<i>0.612588</i>	<i>0.393163</i>	<i>0.1793</i>	<i>0.122457</i>	<i>0.239435</i>
		<i>0.426629</i>	<i>0.540713</i>	<i>0.543239</i>	<i>0.1697</i>	<i>0.200536</i>	<i>0.344748</i>

**Table 3: Accuracy/Diagnostic Statistics for the Parameterized Expectations Models**  
(where delta=0)

Network Model						
	beta = .95					
	sigma:					
	0.02	0.02	0.02	0.1	0.1	0.1
	tau (CRRA):					
	0.5	1.5	3	0.5	1.5	3
<b>Statistic</b>						
PE-Error	9.83E-05	3.21E-05	1.9E-06	0.000119	2.79E-05	9.15E-07
D-M Stat	<b>35.46018</b>	7.97538	<b>2.335482</b>	8.00893	<b>1.22818</b>	7.429544
rsqstat	0.020193	0.029879	0.054394	0.076863	0.128776	0.036034
tr2stat	13.6868	11.88655	11.45928	11.4024	11.23764	14.80248
inv/con ratio	9.750116	9.917064	13.27814	107.537	99.13206	55.11607
con vol	0.035924	0.028555	0.021616	0.022159	0.020743	0.044958
Network Model						
	beta = .98					
	sigma:					
	0.02	0.02	0.02	0.1	0.1	0.1
	tau (CRRA):					
	0.5	1.5	3	0.5	1.5	3
<b>Statistic</b>						
PE-Error	1.38E-05	5.6E-06	3.78E-08	1.45E-05	1.89E-06	3.42E-08
D-M Stat	22.43548	13.65853	8.927994	5.550985	8.088755	7.88883
rsqstat	0.006525	0.015741	0.019844	0.009762	0.019465	0.029758
tr2stat	12.16412	8.339538	14.07553	12.62323	13.53097	15.02458
i-c ratio	6.528208	8.682697	8.562627	36.01803	42.55496	57.47086
con vol	0.054664	0.042604	0.0341	0.051255	0.041692	0.03595
Polynomial Model						
	beta = .95					
	sigma:					
	0.02	0.02	0.02	0.1	0.1	0.1
	tau (CRRA):					
	0.5	1.5	3	0.5	1.5	3
<b>Statistic</b>						
PE-Error	0.000335	4.46E-05	2.08E-06	0.00038	0.000167	0.000157
D-M Stat	<b>34.38042</b>	<b>36.12398</b>	<b>25.9647</b>	10.11301	4.549293	24.67081
rsqstat	0.068241	0.070493	0.036268	0.006872	0.049973	0.480993
tr2stat	12.63014	14.09339	15.42446	13.83102	10.54619	13.60119
i-c ratio	5.771259	3.242505	1.93993	17.69706	1.206709	1.035333
con vol	0.056003	0.092003	0.09234	0.248086	1.105078	2.697219
Polynomial Model						
	beta = .98					
	sigma:					
	0.02	0.02	0.02	0.1	0.1	0.1
	tau (CRRA):					
	0.5	1.5	3	0.5	1.5	3
<b>Statistic</b>						
PE-Error	5.54E-05	9.37E-07	1.29E-07	5.3E-05	2.2E-05	3.8E-06
D-M Stat	<b>30.76413</b>	21.22715	<b>30.99575</b>	10.59343	21.60272	8.016683
rsqstat	0.347499	0.328332	0.038217	0.008772	0.010891	0.027975
tr2stat	9.768334	11.95525	14.26685	16.12978	14.15977	15.24162
i-c ratio	5.73774	46.5541	3.355877	19.56364	2.446527	16.92518
con vol	0.054554	0.016577	0.092944	0.218839	0.706217	0.261653

**Table 4: Neural Network Coefficients and Standard Deviations (model with delta=0)**  
*(Standard deviations in italics, first row based on ga estimates only)*

beta	sigma	tau	constant	Neuron 1		Neuron 2		
				coefficient	k(-1)	theta	k(-1)	theta
0.95	0.02	0.5	1.138065	-0.897447	0.00078	0.267371	-0.040393	3.334775
			<i>0.392197</i>	<i>6.42026</i>	<i>1.249794</i>	<i>3.013832</i>	<i>0.256437</i>	<i>1.802603</i>
			<i>0.435895</i>	<i>6.478718</i>	<i>1.24806</i>	<i>3.218217</i>	<i>0.259761</i>	<i>1.502091</i>
0.95	0.02	1.5	-0.937592	10.52024	-0.030908	0.600233	2.691383	-0.86945
			<i>0.732156</i>	<i>7.3524</i>	<i>0.069582</i>	<i>2.85507</i>	<i>1.198284</i>	<i>2.543489</i>
			<i>0.731495</i>	<i>7.354348</i>	<i>0.066889</i>	<i>2.855128</i>	<i>1.196732</i>	<i>2.543477</i>
0.95	0.02	3	0.005185	4.282718	-0.045222	-0.198397	-0.120871	5.346404
			<i>0.499293</i>	<i>8.397003</i>	<i>1.123835</i>	<i>2.228781</i>	<i>0.031853</i>	<i>2.287282</i>
			<i>0.499286</i>	<i>8.397011</i>	<i>1.123836</i>	<i>2.228781</i>	<i>0.031853</i>	<i>2.287282</i>
0.95	0.1	0.5	-0.589684	0.92521	3.494438	-1.747494	-0.003883	0.169742
			<i>0.400764</i>	<i>6.137187</i>	<i>1.56633</i>	<i>0.72619</i>	<i>1.001637</i>	<i>1.126065</i>
			<i>0.36479</i>	<i>6.143289</i>	<i>1.566407</i>	<i>0.732627</i>	<i>1.000979</i>	<i>1.163954</i>
0.95	0.1	1.5	0.487773	-0.44134	0.054426	-1.615615	-0.01257	-0.0902
			<i>0.979004</i>	<i>10.33687</i>	<i>1.741867</i>	<i>0.724911</i>	<i>1.86919</i>	<i>1.260007</i>
			<i>0.978497</i>	<i>10.33675</i>	<i>1.741795</i>	<i>0.724915</i>	<i>1.86783</i>	<i>1.259879</i>
0.95	0.1	3	0.003629	19.79459	-0.061912	-0.149368	-0.097932	2.381323
			<i>0.105928</i>	<i>9.06452</i>	<i>0.985442</i>	<i>2.682097</i>	<i>0.894234</i>	<i>0.987719</i>
			<i>0.10598</i>	<i>9.064527</i>	<i>0.985437</i>	<i>2.682097</i>	<i>0.894229</i>	<i>0.987719</i>
0.98	0.02	0.5	0.294377	4.856039	-0.006184	0.089262	-0.001252	-0.743309
			<i>0.023587</i>	<i>3.156921</i>	<i>1.342268</i>	<i>0.917426</i>	<i>0.002353</i>	<i>0.764437</i>
			<i>0.023577</i>	<i>3.156921</i>	<i>1.341903</i>	<i>0.917426</i>	<i>0.002069</i>	<i>0.764435</i>
0.98	0.02	1.5	-0.857313	-0.098258	0.013748	-4.528765	2.669067	1.947095
			<i>0.422881</i>	<i>4.234846</i>	<i>1.169247</i>	<i>1.860844</i>	<i>1.607114</i>	<i>0.991459</i>
			<i>0.422861</i>	<i>4.234854</i>	<i>1.169241</i>	<i>1.860844</i>	<i>1.607111</i>	<i>0.991459</i>
0.98	0.02	3	0.002687	4.748672	-0.024321	-0.052801	-0.133511	7.582655
			<i>0.39384</i>	<i>2.979929</i>	<i>0.687991</i>	<i>2.068783</i>	<i>0.480554</i>	<i>3.332521</i>
			<i>0.39384</i>	<i>2.97994</i>	<i>0.687991</i>	<i>2.068783</i>	<i>0.480554</i>	<i>3.332521</i>
0.98	0.1	0.5	0.315984	5.920693	-0.004606	-0.026632	-1.471257	0.033729
			<i>0.44757</i>	<i>4.329235</i>	<i>1.596258</i>	<i>0.731452</i>	<i>0.755224</i>	<i>0.773969</i>
			<i>0.447553</i>	<i>4.329235</i>	<i>1.596258</i>	<i>0.731452</i>	<i>0.755217</i>	<i>0.773969</i>
0.98	0.1	1.5	0.039613	4.123392	-0.012619	-0.095063	-0.010242	0.114006
			<i>0.2644</i>	<i>4.212807</i>	<i>1.708914</i>	<i>3.011417</i>	<i>0.034417</i>	<i>2.776134</i>
			<i>0.264392</i>	<i>4.212824</i>	<i>1.708912</i>	<i>3.011416</i>	<i>0.034431</i>	<i>2.776134</i>
0.98	0.1	3	-0.99755	4.330734	-0.026375	-0.072781	0.684483	-0.108325
			<i>0.726231</i>	<i>8.348796</i>	<i>0.081674</i>	<i>5.260965</i>	<i>1.214151</i>	<i>0.703686</i>
			<i>0.726232</i>	<i>8.348804</i>	<i>0.081673</i>	<i>5.260965</i>	<i>1.214151</i>	<i>0.703686</i>

**Table 5: Polynomial Coefficients and Standard Deviations (model with delta=0)**  
*(Standard deviations in italics, first row based on ga estimates only)*

beta	sigma	tau	constant	First-Degree		Second-Degree		Cross-term
				ln[k(-1)]	ln(theta)	ln[k(-1)]	ln(theta)	ln[k(-1)]ln(theta)
0.95	0.02	0.5	1.049333	0.169402	0.862621	-0.12054	-0.026686	-0.036381286
			<i>0.566708</i>	<i>0.568728</i>	<i>0.676591</i>	<i>0.10204</i>	<i>0.018142</i>	<i>0.081210085</i>
			<i>0.566705</i>	<i>0.568739</i>	<i>0.676597</i>	<i>0.101987</i>	<i>0.018142</i>	<i>0.08138375</i>
0.95	0.02	1.5	0.47191	0.841224	0.00468	-0.004435	0.001858	-0.177642703
			<i>1.292867</i>	<i>1.116388</i>	<i>1.006154</i>	<i>0.194771</i>	<i>0.011638</i>	<i>0.186773802</i>
			<i>1.292868</i>	<i>1.116388</i>	<i>1.006154</i>	<i>0.194765</i>	<i>0.011638</i>	<i>0.186790414</i>
0.95	0.02	3	0.649016	0.046461	2.167939	-0.486089	0.014307	-0.042049177
			<i>0.95692</i>	<i>0.353137</i>	<i>1.555847</i>	<i>0.347877</i>	<i>1.38929</i>	<i>0.055872158</i>
			<i>0.956929</i>	<i>0.353129</i>	<i>1.555845</i>	<i>0.347879</i>	<i>1.384274</i>	<i>0.055890782</i>
0.95	0.1	0.5	-0.812806	1.071377	0.256924	-0.038341	-0.003047	-0.131405318
			<i>0.581749</i>	<i>0.249445</i>	<i>0.389951</i>	<i>0.057267</i>	<i>0.001099</i>	<i>0.027636859</i>
			<i>0.581745</i>	<i>0.249416</i>	<i>0.389957</i>	<i>0.05715</i>	<i>0.001063</i>	<i>0.027700824</i>
0.95	0.1	1.5	-0.771198	0.459113	1.313694	-0.178094	-0.013663	-0.047291292
			<i>0.85218</i>	<i>0.292137</i>	<i>0.581201</i>	<i>0.058952</i>	<i>0.054098</i>	<i>0.036016623</i>
			<i>0.843823</i>	<i>0.28336</i>	<i>0.577748</i>	<i>0.08333</i>	<i>0.007123</i>	<i>0.031801595</i>
0.95	0.1	3	0.449585	-0.029935	0.651348	-0.062844	-0.016365	-0.001049174
			<i>0.969351</i>	<i>0.807533</i>	<i>1.270668</i>	<i>0.151778</i>	<i>0.114722</i>	<i>0.077112426</i>
			<i>1.054116</i>	<i>0.801129</i>	<i>0.727554</i>	<i>0.10201</i>	<i>0.1139</i>	<i>0.076565707</i>
0.98	0.02	0.5	0.988052	0.460422	-0.199462	0.027416	0.003972	-0.076378012
			<i>0.904092</i>	<i>0.447772</i>	<i>0.750289</i>	<i>0.111879</i>	<i>0.005829</i>	<i>0.06810283</i>
			<i>0.904092</i>	<i>0.447773</i>	<i>0.750289</i>	<i>0.111876</i>	<i>0.005829</i>	<i>0.068094498</i>
0.98	0.02	1.5	0.397277	-0.008409	-0.051041	0.007418	0.001856	-0.006468998
			<i>0.500495</i>	<i>0.523949</i>	<i>0.88182</i>	<i>0.150735</i>	<i>0.004225</i>	<i>0.079859365</i>
			<i>0.500495</i>	<i>0.523949</i>	<i>0.881819</i>	<i>0.150736</i>	<i>0.004225</i>	<i>0.079872828</i>
0.98	0.02	3	0.315705	0.020236	0.115688	-0.022289	8.11E-05	-0.014866692
			<i>2.384722</i>	<i>1.011237</i>	<i>1.168887</i>	<i>0.122304</i>	<i>0.581603</i>	<i>0.104644636</i>
			<i>2.384807</i>	<i>1.010995</i>	<i>1.168771</i>	<i>0.122497</i>	<i>0.587446</i>	<i>0.104713587</i>
0.98	0.1	0.5	-0.543917	0.918745	0.160874	-0.02284	-0.001889	-0.11037633
			<i>0.764203</i>	<i>0.327475</i>	<i>0.072465</i>	<i>0.010313</i>	<i>0.00113</i>	<i>0.045111908</i>
			<i>0.764202</i>	<i>0.32747</i>	<i>0.072465</i>	<i>0.0103</i>	<i>0.001124</i>	<i>0.045120704</i>
0.98	0.1	1.5	1.043739	0.720442	-0.891994	0.146776	-0.011026	-0.146950375
			<i>0.928105</i>	<i>0.385209</i>	<i>1.204833</i>	<i>0.182197</i>	<i>0.017014</i>	<i>0.063599101</i>
			<i>0.928382</i>	<i>0.38439</i>	<i>1.206208</i>	<i>0.17308</i>	<i>0.016771</i>	<i>0.05856538</i>
0.98	0.1	3	0.496401	-0.129347	0.108197	-0.011933	0.011442	0.008598994
			<i>1.483095</i>	<i>0.482269</i>	<i>1.73272</i>	<i>0.179923</i>	<i>0.057963</i>	<i>0.032327403</i>
			<i>1.488434</i>	<i>0.484959</i>	<i>1.756383</i>	<i>0.183092</i>	<i>0.0595</i>	<i>0.032631159</i>

**Table 6: Comparison with Alternative Methods under Low Variance Assumption:  $\sigma = .02$**

	Panel A: beta = .95 <u>tau = .5</u>				Panel B: beta = .95 <u>tau = 1.5</u>				Panel C: beta = .95 <u>tau = 3</u>			
	<i>D-M Stat</i>	<i>TR-Stat</i>	<i>R-SQ</i>	<i>IC Ratio</i>	<i>D-M Stat</i>	<i>TR-Stat</i>	<i>R-SQ</i>	<i>IC Ratio</i>	<i>D-M Stat</i>	<i>TR-Stat</i>	<i>R-SQ</i>	<i>IC Ratio</i>
<i>Duffy/McNelis-NN</i>	35.46	13.69	0.02	9.75	7.98	11.89	0.03	9.92	2.34	11.46	0.05	13.28
<i>Duffy/McNelis-PA</i>	34.38	12.63	0.07	5.77	36.12	14.09	0.07	3.24	25.96	15.42	0.04	1.94
<i>Christiano-Loq LQ</i>	17.00	10.00	0.43	29.00	10.00	10.00	0.05	11.00	18.00	19.00	0.02	8.00
<i>Ingram</i>	10.00	17.00	0.44	30.00	11.00	165.00	0.06	12.00	12.00	394.00	0.03	20.00
<i>Den Haan/Marcel</i>	18.00	15.00	0.42	30.00	18.00	14.00	0.06	13.00	12.00	13.00	0.03	10.00
<i>McGratten</i>	96.00	19.00	0.34	24.00	22.00	19.00	0.04	9.00	17.00	19.00	0.02	7.00
<i>Sims</i>	12.00	24.00	0.44	31.00	12.00	24.00	0.07	13.00	12.00	22.00	0.04	11.00
<i>Tauchen</i>	704.00	11.00	0.50	3.00	558.00	9.00	0.38	2.00	502.00	14.00	0.33	2.00
	Panel D: beta = .98 <u>tau = 0.5</u>				Panel E: beta = .98 <u>tau = 1.5</u>				Panel F: beta = .98 <u>tau = 3</u>			
	<i>D-M Stat</i>	<i>TR-Stat</i>	<i>R-SQ</i>	<i>IC Ratio</i>	<i>D-M Stat</i>	<i>TR-Stat</i>	<i>R-SQ</i>	<i>IC Ratio</i>	<i>D-M Stat</i>	<i>TR-Stat</i>	<i>R-SQ</i>	<i>IC Ratio</i>
<i>Duffy/McNelis-NN</i>	22.44	12.16	0.01	6.53	13.66	8.34	0.02	8.68	8.93	14.08	0.02	8.56
<i>Duffy/McNelis-PA</i>	30.76	9.77	0.35	5.74	21.23	11.96	0.33	46.55	31.00	14.27	0.04	3.36
<i>Christiano-Loq LQ</i>	28.00	20.00	0.24	132.00	16.00	25.00	0.03	59.00	12.00	16.00	0.01	45.00
<i>Ingram</i>	8.00	15.00	0.33	162.00	11.00	203.00	0.04	66.00	12.00	381.00	0.02	98.00
<i>Den Haan/Marcel</i>	30.00	15.00	0.35	178.00	7.00	14.00	0.04	78.00	9.00	14.00	0.02	74.00
<i>McGratten</i>	62.00	19.00	0.21	112.00	26.00	17.00	0.02	44.00	21.00	16.00	0.01	38.00
<i>Sims</i>	11.00	19.00	0.36	171.00	12.00	16.00	0.04	66.00	10.00	14.00	0.02	59.00
<i>Tauchen</i>	322.00	16.00	0.34	2.00	234.00	13.00	0.27	2.00	215.00	10.00	0.27	2.00

**Table 7: Comparison with Alternative Methods under High Variance Assumption:  
sigma=.10**

	Panel A: beta = .95				Panel B: beta = .95			
	tau = .5				tau = 1.5			
	D-M Stat	TR-Stat	R-SQ	I/C Ratio	D-M Stat	TR-Stat	R-SQ	I/C Ratio
Duffy/McNelis-NN	8.01	11.40	0.08	107.54	1.23	11.24	0.13	99.13
Duffy/McNelis-PA	10.11	13.83	0.01	17.70	4.55	10.55	0.05	1.21
Christiano-Loq LQ	56.00	31.00	0.36	24.00	32.00	16.00	0.04	9.00
Ingram	11.00	46.00	0.35	29.00	8.00	123.00	0.18	170.00
Den Haan/Marcet	27.00	12.00	0.40	28.00	22.00	11.00	0.04	9.00
McGratten	84.00	21.00	0.13	6.00	55.00	20.00	0.04	3.00
Sims	11.00	27.00	0.41	30.00	14.00	26.00	0.06	12.00
Tauchen	584.00	8.00	0.50	3.00	396.00	9.00	0.38	2.00
	Panel C: beta = .98				Panel D: beta = .98			
	tau =0.5				tau=1.5			
	D-M Stat	TR-Stat	R-SQ	I/C Ratio	D-M Stat	TR-Stat	R-SQ	I/C Ratio
Duffy/McNelis-NN	5.55	12.62	0.01	36.02	8.09	13.53	0.02	42.55
Duffy/McNelis-PA	10.59	16.13	0.01	19.56	21.60	14.16	0.01	2.45
Christiano-Loq LQ	34.00	12.00	0.12	45.00	44.00	25.00	0.04	36.00
Ingram	11.00	75.00	0.26	155.00	10.00	172.00	0.10	490.00
Den Haan/Marcet	22.00	11.00	0.33	168.00	25.00	12.00	0.03	55.00
McGratten	69.00	18.00	0.07	17.00	22.00	14.00	0.04	10.00
Sims	12.00	22.00	0.32	165.00	12.00	19.00	0.04	64.00
Tauchen	284.00	19.00	0.34	2.00	153.00	12.00	0.27	2.00

**Figure 1:  
e(h) Statistic for Network (n) and Polynomial (p)  
Methods for Two Values of Beta**

