Feedback Approximation of the Stochastic Growth Model by Genetic Neural Networks*

Sibel Sirakaya[a] Stephen Turnovsky[b,†] M. Nedim Alemdar[c]

[a] *Departments of Economics and Statistics, and CSSS, University of Washington, Seattle, WA 98195*

[b] *Department of Economics, University of Washington, Seattle, WA 98195*

[c] *Department of Economics, Bilkent University, 06800 Bilkent, Ankara, Turkey*

Abstract

A direct numerical optimization method is developed to approximate the one-sector stochastic growth model. The feedback investment policy is parameterized as a neural network and trained by a genetic algorithm to maximize the utility functional over the space of time-invariant investment policies. To eliminate the dependence of training on the initial conditions, at any generation, the same stationary investment policy (the same network) is used to repeatedly solve the problem from differing initial conditions. The fitness of a given policy rule is then computed as the sum of payoffs over all initial conditions. The algorithm performs quite well under a wide set of parameters. Given the general purpose nature of the method, the flexibility of neural network parametrization and the global nature of the genetic algorithm search, it can be easily extended to tackle problems with higher dimensional nonlinearities, state spaces and/or discontinuities.

Key Words: Stochastic growth model, Genetic algorithms, Neural networks

*We gratefully acknowledge the constructive comments of two anonymous referees.

†Corresponding author. Department of Economics, University of Washington, Seattle 98195, WA. E-mail: sturn@u.washington.edu.

1 Introduction

The Ramsey growth model continues to be one of the main analytical tools of modern growth theory. Despite the insights provided by the endogenous growth model, pioneered by Romer (1986), adaptations of the Ramsey model have generally been superior in reconciling the theory with empirical evidence; see e.g. Jones (1995). Much of the focus of recent growth theory has been on the formulation and extension to stochastic economies. This is particularly so in the real business cycle literature, where the emphasis has been on characterizing the nature of the short-run stochastic properties, such as the variance and covariances among key economic variables; see e.g. Cooley (1995). But the effect of uncertainty on the long-run evolution of the economy is also important and has a long tradition, dating back to the early seminal work of Mirrlees (1965), Brock and Mirman (1972), and others.

One of the problems with the nonlinear stochastic growth model is that closed form solutions can be obtained only in very special cases.[1] Accordingly, the solution procedures have inevitably relied on approximation and numerical methods. Early work by King, Plosser, and Rebelo (1988) employ linear approximations. While these may be adequate for understanding certain aspects of the equilibrium, they are generally not well suited for handling questions pertaining to welfare comparisons for which second order approximations become necessary; see e.g. Judd (1998), Schmitt-Grohé and Uribe

[1] One case where closed form solutions can be conveniently obtained is if the technology is of the "AK" form associated with the endogenous growth model; see Eaton (1981) for an early example and Turnovsky (2000) for a wider range of examples that allow for richer underlying economic structures.

(2004), and Kim and Kim (2005).

Perturbation methods in general are easy to implement and do quite well locally when no binding constraints are present. However, since the technique depends on a Taylor series expansion around the steady state, it may perform poorly away from the equilibrium. Further, for high dimensional nonlinear problems accurate derivatives (analytical or numerical) may not be readily available making the method less dependable.

Another indirect numerical approach to solving the stochastic growth problem is to use the methods of functional approximation to obtain a policy (value) function that best fits the Euler (Bellman) equation in some average sense. Weighted residual methods use a linear combination of known basis functions, which are usually polynomials, to approximate the policy (value) function over the collocation nodes. That is, a root finding routine recovers the coefficients on each basis function that globally best fits the Euler (Bellman) equation; see e.g. Judd (1998) and Miranda and Fackler (2002). Weighted residual methods perform well within the domain of approximation and require modest computational effort. To increase the numerical accuracy one may increase the order of the polynomial and/or the domain of approximation with the attendant computational costs. As the state space gets large, however, the cost may become prohibitive.

We propose a new direct method that can efficiently provide highly accurate approximations to the solution of the stochastic growth model. In contrast to the existing numerical methods, which rely upon either Euler or Bellman equations to approximate the solution, we adopt a direct numerical

optimization approach that requires neither. Specifically, we first parameterize the policy function by a feedforward neural network which is then trained by a genetic algorithm to search over all time-invariant strategies so as to maximize the objective functional subject to the resource and non-negativity constraints. This is an *on-line*, general purpose algorithm which only requires the user to supply the objective functional and the constraints so that the computational effort on the part of the user is minimal.

The neural network specification offers several important advantages over the traditional numerical techniques, such as spline functions or radial basis functions. First, feedforward neural networks have proved to be universal function approximators. Under general regularity conditions, a sufficiently complex single hidden layer feedforward neural network can approximate any member of a class of function to any degree of accuracy.[2] Second, nonlinearly parameterized nature of feedforward neural networks allow them to use fewer parameters to achieve the same degree of approximation accuracy as opposed to linearly parameterized techniques which require an exponential increase in the number of parameters. Third, neural networks with a sigmoid activation function at the output layer naturally deliver control bounds, while such bounds constitute a major problem for linearly parameterized techniques. Fourth, neural networks can easily be applied to problems that admit bang-bang solutions, while this constitutes a major difficulty for other conventional numerical solution methods.

There are several good reasons for our choice of genetic algorithms (GA)

[2]See for example, Funahashi (1988) and Hornik, Stinchcombe and White (1989).

to train the neural networks as well. First, in contrast to the gradient-descent methods, genetic implementations do not use the gradient information. Thus, they do not require the continuity and the existence of derivatives of the objective functionals and state transition functions. The only restrictions are that they be bounded, a natural consequence of which is that our method can be applied to a larger class of problems. Second, GAs are global search algorithms that start completely blind and learn gradually. Regardless of the initial parameter values, they ensure converge to an approximate global optimum by exploiting the domain space and relatively better solutions through genetic operators. Gradient-descent methods, on the other hand, need gradient information and may get stuck in a local optimum or fail to converge at all, depending on the initial parameter values.

GAs are not without difficulty either. First, the so-called competing conventions problems may arise since structurally different networks can be functionally equivalent. Genetic algorithms operate on genotypes which represent a network structure. Consequently, structurally different networks are represented by different genotypes. If some structures are functionally equivalent, then the crossover operator may degenerate the search by creating inferior offsprings. Specifically, the farther apart are the weights of a node and nodes of different layers located on a chromosome, the more likely it is for the standard one-point crossover operator to disrupt them. Thus, placing all incoming weights of a node and all nodes side by side may help resolve this problem. Also, a more assiduous use of the crossover operator, together with more aggressive rank selection and mutation, are also shown

to be useful (Branke, 1995). Finally, GAs may be computationally more time consuming as compared with the conventional gradient-descent techniques. But despite these potential difficulties, we wish to emphasize the contribution of our method in that it can solve problems that may otherwise be analytically and computationally intractable.

In executing the algorithm, we first parameterize the policy function by the weights of a suitable neural network, thereby transforming the search space from a set of rules to a set of neural net weights. Next, an artificially intelligent trainer, a GA, is assigned to breed fitter weights for the network. In any given generation, the algorithm starts from multiple sets of initial states. Since, we are searching for the stationary policy rules, the same set of neural net weights are used to compute the payoffs. The raw fitness of a policy rule in the GA population is calculated as the sum of all payoffs across all initial states. Our rationale for summing over a set of initial states is two-fold: to avoid the dependence of the weight training on the initial conditions and also to speed up GA learning.

When our algorithm is applied to the standard stochastic growth problem, it performs quite well under many measures of accuracy. For instance, when exact solutions are available, our solution method provides approximations that are highly accurate under various error measures. When analytic solutions are not known, our approximation method, in general, produce modest Euler residual errors. Compared with weighted residual methods that iterate on Euler equation to recover the optimal policy, such as collocation, these errors may seem larger. However, the magnitudes must be judged in the light

of the fact that the state space the genetically evolved neural networks search is many times larger than the domain of approximation for the collocation methods. Hence, the proposed search strategy is more robust. In contrast with the numerical methods reported in Taylor and Uhlig (1990) and Duffy and McNelis (2001), only our genetically evolved neural networks produce solutions that pass tests of the martingale difference property of both the Euler equation residuals and the productivity shocks, and are also consistent with the random walk behavior of consumption. Given that our method is a "direct" approach with minimal off-line computing effort, these results suggest that our method can complement the existing methods of approximation resulting in substantial computational gains in problems where the search terrain is highly erratic or unknown due to discontinuities, nonlinearities or large state spaces.

The balance of the paper is as follows. In section 2, we briefly discuss the stochastic growth model. Section 3 first presents a short overview of the neural networks and genetic algorithms, and then proceeds to show how genetic neural networks can approximate the solution to the stochastic growth model. Section 4 discusses the results of simulations for two versions of the model, one in which an analytical solution for the model is available and another version in which there is no closed-form solution. Conclusions and possible extensions follow.

2 The Model

Consider the following one-sector stochastic growth model in which agents are assumed to be infinitely lived and to solve the following problem:

$$\max_{\{i_t\}_{t=0}^{\infty}} J = E_0 \sum_{t=0}^{\infty} \beta^t U(c_t) = E_0 \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\tau}}{1-\tau},\ 0 < \beta < 1,\ \tau > 0, \tag{1}$$

subject to

$$c_t + i_t = y_t,$$

$$y_t = \theta_t k_t^{\alpha},$$

$$i_t = k_{t+1} - (1-\delta)k_t,\ k_{t+1},\ c_t > 0\ \forall t \text{ and } k_0 \text{ given},$$

where c_t, i_t, y_t and k_t respectively, are consumption, investment, production and capital stock at time t.[3] The constant time discount factor is β, and δ denotes the constant rate of capital depreciation. The stochastic technology shock is denoted by θ_t and is assumed to evolve according to

$$\ln(\theta_t) = \rho \ln(\theta_{t-1}) + \varepsilon_t,$$

[3]Note that our problem formulation is a bit nonstandard as we are looking for the maximum over investment policies rather than consumption. We wish to remain consistent in our notation since we later parameterize investment policy as a neural network.

with $|\rho| < 1$, and the error process $\varepsilon_t \sim N(0, \sigma)$. Stationary optimal investment policies obey the following Euler equation

$$(\theta_t k_t^{\alpha} - i_t)^{-\tau} = E_t[\beta(\theta_{t+1} k_{t+1}^{\alpha}, -i_{t+1})^{-\tau}(\alpha\theta_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta)], \tag{2}$$

as well as the model constraints. That is, the solution is a time invariant investment policy,

$$i_t = h(\theta_t, k_t),$$

and a law of motion for the stock of capital,

$$k_{t+1} = g(\theta_t, k_t) = h(\theta_t, k_t) + (1 - \delta)k_t.$$

Problem (1) has a closed-form analytical solution only when $\tau = \delta = 1$, i.e., preferences are logarithmic and the capital stock fully depreciates every period. This special case is studied in detail in Brock and Mirman (1972). In this case, the optimal investment policy and the corresponding path of the optimal capital stock are given by (see, e.g., Sargent, 1987, p. 122):

$$i_t = k_{t+1} = \alpha\beta\theta_t k_t^{\alpha}.$$

For all other cases, numerical approximation methods are needed to solve for the optimal investment policy and the capital accumulation path. The first approach uses the stochastic version of the Bellman equation to iterate

on the value function to recover the optimal ivestment policy. Towards that, the continuous shocks and the states in the original problem are discretized to evaluate the expectation numerically and to solve the resulting discrete problem over a grid of points. By refining the grid, an arbitrary level of approximation accuracy can be achieved (see for example Christiano, 1990; Tauchen, 1990).

A large number of the existing methods focus on the Euler equation, Eq.(2), together with model constraints for numerical computation of the stochastic growth model. These methods differ with regard to how they handle the nonlinear expectation in (2). One approach replaces the expectation in (2) by realized future values. This is equivalent to an assumption of perfect foresight. The extended path method of Fair and Taylor (1983), implemented for the stochastic growth model by Gagnon (1990), is an example of this method.

An alternate approach is to approximate the original problem by a simpler version for which a closed form solution is readily available. The log-linearization of the model and the method of parameterized expectations of den Haan and Marcet (1990) are examples of this procedure. The method of parameterized expectations explicitly approximates the nonlinear expectation in the Euler equation by a known functional form, the parameters of which can be estimated from realizations of the model. Den Haan and Marcet (1990,1994) use non-linear least squares, while Christiano and Fisher (1997) use ordinary least squares to estimate the parameters. Duffy and McNelis (2001), on the other hand, parameterize the expectation function by

neural networks and use genetic algorithms to initialize gradient searches for the network weights.

Another commonly used strand of the numerical approach is the so-called weighted residual methods. First, the policy (value) function is represented as a linear combination of known basis functions, which are typically polynomials. The coefficients on each basis function are obtained by requiring the approximant to satisfy the functional equation, not at all possible points of the domain, but rather at a number of prescribed points. Collocation, least-squares and Galerkin methods are the most commonly used weighted residual methods. A thorough treatment of these techniques can be found in Fletcher (1984), Judd (1998), McGrattan (1999) and Miranda and Fackler (2002). Galerkin and collocation projection methods have been shown to be very successful in the approximation of the standard stochastic growth model; see Judd (1992, 1998). In particular, Chebyshev polynomials have been shown to provide very accurate approximations of the policy function in many examples; see e.g. Judd (1998) or Heer and Maussner (2005).

Weighted residual methods are, however, not free from difficulties. First, even if they are less prone to the curse of dimensionality, they do eventually suffer from it. Second, polynomial and spline approximants may perform poorly outside the domain of approximation. Furthermore, given the stochastic nature of the problem, it is possible for the solution algorithm to run into states outside the bounds early on thereby creating convergence problems. Moreover, if nondifferentiabilities exist, this will undermine the rootfinding algorithm used to compute the optimum action at each state

node (Miranda and Fackler, 2002).

We depart from the existing indirect methods which rely either on the Bellman or the Euler equation. In order to exploit the efficiency of direct numerical optimization and also take advantage of robust global search, we propose to parameterize the investment policy by a feedforward neural network, and then use genetic algorithms to search over all time-invariant strategies so as to optimize the objective functional subject to the constraints. The next section describes the details of our algorithm.

3 A Brief Note on Neural Networks and Genetic Algorithms

3.1 Neural Networks

Neural networks are information-processing paradigms that mimic highly interconnected, parallely structured biological neurons. They are trained to learn and generalize from a given set of examples by adjusting the synaptic weights between the neurons.[4]

Consider an L layer (or $L-1$ hidden layer) feedforward neural network, with the input vector $z^0 \in \mathcal{R}^{r_0}$ and the output vector $\phi(z^0) = z^L \in \mathcal{R}^{r_L}$. As in Narenda and Parthasarthy (1990), we refer to this class of networks as

[4]For the sake of compactness, the notation in this section closely follows that of Narenda and Parthasarthy (1990). A well documented theory of neural networks can be found in Hecht-Nielsen (1990) and Hertz, Krogh and Palmer (1991).

$\mathcal{N}^L_{r_0,r_1,\ldots,r_L}$. The recursive input-output relationship is given by

$$y^j = w^j z^{j-1} + v^j, \tag{3}$$

$$z^j = \hat{\psi}_j(y^j) = (\psi_j(y^j_1)\ \psi_j(y^j_2)\ \ldots\ \psi_j(y^j_{r_j}))', \tag{4}$$

where $w^j \in \mathcal{R}^{r_j \times r_{j-1}}$ and $v^j \in \mathcal{R}^{r_j}$ for $j = 1, 2, \ldots, L$ are the connection and the bias weights respectively. The dimension of y^j and z^j is denoted by r_j. The scalar activation functions, $\psi_j(.)$ are usually sigmoids, e.g. $\psi_j(.)$ =tanh(.) or $\psi_j(.) = 1/(1 + exp(-(.))$ in the hidden layers. At the output layer, the activation functions, $\psi_L(.)$, can be linear, e.g. $\psi_L(.) = (.)$, if the outputs have no natural bounds. If, however, they are bounded by $\gamma_{min} \leq z^L \leq \gamma_{max}$, then one may choose:

$$\psi_L(.) = \gamma_{min} + \frac{\gamma_{max} - \gamma_{min}}{1 + exp(-(.))}. \tag{5}$$

Letting $\omega = (w^1\ v^1\ \ldots\ w^L\ v^L)$, the approximating function has the general representation:

$$\phi(z^0, \omega) = \hat{\psi}_L(w^L \hat{\psi}_{L-1}(w^{L-1} \hat{\psi}_{L-2}(\ldots(w^2 \hat{\psi}_1(w^1 z^0 + v^1) + v^2) + v^3) + \ldots + v^{L-1}) + v^L). \tag{6}$$

3.2 Genetic Algorithms

The neural network parameterizing the policy function in our algorithm is trained by a genetic algorithm. That is, the interconnection weights between the neurons are incrementally adjusted by a GA to optimize the objective functional subject to the constraints. A basic GA consists of iterative procedures, called generations. In each generation, say s, a GA maintains a constant size population, Pop(s), of candidate solution vectors to the problem at hand. Each individual in Pop(s) is coded as a finite-length string, usually over the binary alphabet ($\{0, 1\}$). The initial population, Pop(0), is generally random.

At any generation, each individual in a population is assigned a 'fitness score' depending on how good a solution it is relative to the population. During a single reproduction phase, relatively fit individuals are selected from a pool of candidates some of which are recombined to generate a new generation. Better solutions breed faster while bad solutions vanish. Basic recombination operators are *mutation* and *crossover*.

Crossover randomly chooses two members ('parents') from the population, then creates two similar off-springs by swapping the corresponding segments of the parents. Crossover can be considered as a way of further exploration by exchanging information between two potential solutions. Mutation randomly alters single bits of the bit strings encoding individuals with a probability equal to the mutation rate *pmut*. It can be interpreted as experimenting to breed fitter solutions.

GAs are highly parallel mathematical structures. While they operate on individuals in a population, they collect and process vast amounts of information by exploiting the similarities in classes of individuals, which Holland calls *schemata*. These similarities in classes of individuals are defined by the lengths of common segments of bit strings. By operating on n individuals in one generation, a GA collects information approximately about n^3 individuals (Holland, 1975).

Parallelism can be explicit as well in the sense that more than one GA can generate and collect data independently and that genetic operators may be implemented in parallel (See Mühlenbein, 1992). Parallel genetic algorithms are inspired by the biological evolution of species in isolated locales. To mimic this evolutionary process, a population is divided into subpopulations and a processor is assigned to each to separately apply genetic operators while allowing for periodic communication between them. Subpopulations, specialize on one portion of the problem and communicate among themselves to learn about the remainder. This idea is exploited in a number of studies to approximate the equilibria in deterministic dynamic games and can be easily extended to stochastic environments.[5]

[5] Alemdar and Özyıldırım (1998) use parallel GAs to approximate open-loop Nash equilibrium in differential games. Sirakaya and Alemdar (2003) employ parallel GAs to approximate feedback Nash equilibria in deterministic dynamic games.

3.3 Approximation of the Policy Function with Genetic Neural Networks

We first parameterize the policy rule by a neural network as

$$h(k_t, \theta_t) = \phi(z_t^0, \omega). \tag{7}$$

where ω is the vector of connection and bias weights of the network approximating the policy function and z_t^0 is the time t input vector. The time t input vector to the network is an r_0-dimensional vector of the state variables at time t, such as $z_t^0 = (k_t \ \theta_t)'$ or $z_t^0 = (k_t \ k_t, \theta_t)'$. The time $t+1$ input to the network, z_{t+1}^0, is generated as follows. First, given θ_0, we generate a single draw of a series for θ_t of length T. This series is drawn only once and repeatedly used in finding a solution. Next, to get k_{t+1}, we use

$$k_{t+1} = \phi(z_t^0, \omega) + (1-\delta)k_t \ , k_0 \text{ given.}$$

Note that given the initial states, the search space is now transformed from a set of rules to a set of neural net weights. The ability of the trained network to generalize, however, will be limited as the training depends upon the initial conditions of the problem.

Thus, our next task is to devise a method so that the trained network can generalize over a wider set of initial conditions. Towards that, we note that if a stationary policy rule maximizes the representative agent's expected

lifetime utility for any given initial states, then it must also maximize the sum of his expected lifetime utilities over a set of initial states. Denoting a set of initial states as Π, for any given set of weights and initial states $(k_0, \theta_0) \in \Pi$, we can generate the sample paths for the states (thus the input vectors at any time t) and the policies as described above. Thus, the sum of utilities over all initial states becomes

$$\tilde{J}(\omega) = \sum_{(k_0,\ \theta_0)\in\Pi} \sum_{t=0}^{T} \beta^t \ \frac{(y_t - \phi(z_t^0, \omega))^{1-\tau}}{1-\tau}. \tag{8}$$

Hence, if the neural nets approximating the stationary feedback policies are trained to maximize this sum, then they will have a better generalizing capacity.

In passing, we note that many neural networks can parameterize the feedback rule $h(k_t, \theta_t)$. There exists no hard-and-fast rule of choosing a network architecture other than a systematic trial and error approach. While a network architecture with too many layers and neurons may be very time consuming and may not offer significant improvement over an architecture with fewer layers and neurons, too few layers and neurons may result in poor approximations. As a general rule, simpler architectures are more preferable because they learn faster.

To approximate the policy rule, we use a GA to train the neural network as represented in Eq. (7). At any generation $s \in S$, a GA operates on a constant size population, N, of neural net weights:

$$\text{Pop}(s) = \{\omega_1(s), \omega_2(s), \ldots, \omega_b(s), \ldots, \omega_N(s)\}.$$

where $\omega_b(s) \in \text{Pop}(s)$ represents a vector of potential weights approximating optimal policy rule.[6]

GA evaluates each individual $b \in \text{Pop}(s)$ by computing its raw fitness[7],

$$\tilde{J}(\omega_b(s)) = \sum_{(k_0,\ \theta_0)\in\Pi} \sum_{t=0}^{T} \beta^t \ \frac{(y_t - \phi(z_t^0, \omega_b(s)))^{1-\tau}}{1-\tau}.$$

The search is initialized from a random population, Pop(0). Given a random $d \in \text{Pop}(0)$, GA finds the best performing individual, b, such that

$$\tilde{J}(\omega_b(0)) \geq \tilde{J}(\omega_g(0)),$$

for $g = 1, 2, \ldots, b-1, b+1, \ldots, N$. Next, using the evolutionary operators, a new generation of population is formed from the relatively fit individuals and their fitness scores are recalculated.

The above procedures is repeated for a number of generations. That is, at any generation s, GA proceeds with the search if there exists a b such that:

$$\tilde{J}(\omega_b(s)) \geq \tilde{J}(\omega_g(s)),$$

for $g = 1, 2, \ldots, b-1, b+1, \ldots, N$.

As the search evolves, fitter individuals proliferate, thanks to the reproduction and crossover operators, until $s' \leq S$ whence for any $s \geq s'$ there exists no individuals $b \in \text{Pop}(s)$ such that

[6]We place all incoming weights of a node and all nodes side by side on the chromosomes.

[7]When a string representing the weights of a network results in a rate of disinvestment that is greater than the existing capital stock, it is punished by a high penalty; namely -1000000.

$$\hat{J}(\omega_b(s)) > \tilde{J}(\omega_F),$$

where ω_F are the weights that best approximates the equilibrium policy rule.

The following pseudo code outlines the steps involved in our GA search The general outline of a GA is:

```
procedure GA;
begin
initialize population Pop(0);
evaluate Pop(0);
s=1;
repeat
   select Pop(s) from Pop(s-1);
   recombine Pop(s);
   evaluate Pop(s);
until (termination condition);
```

At this point, a word of caution is in order about the selection operator. The search terrain for the neural network generally is highly nonlinear. Thus, it becomes imperative that a selection procedure be adopted that will sustain the evolutionary pressure. An elitist selection strategy alone will fail on this account. In our simulations, we use *fitness rank selection* together with elitism in selection procedures. With fitness rank selection, individuals are first sorted according to their raw fitness, and then using a linear scale reproductive fitness scores assigned according to their ranking. Rank selection prevents premature convergence since the raw fitness values have no direct

impact on the number of offspring. The individual with the highest fitness may be much superior to the rest of the population or it may be just above the average; in any case, it will expect the same number of offspring. Thus, superior individuals are prevented from taking over the population too early causing a false convergence.

4 Simulation Results

After a round of experimentations, we adopt the following disconnected feed-forward network from $\mathcal{N}^2_{2,2,1}$ to parameterize the policy rule $h(k_t, \theta_t)$:

$$\phi(z_t^0, \omega) = \gamma_{min} + (\gamma_{max} - \gamma_{min}) z_t^2,$$

$$z_t^2 = \hat{\psi}_2(y_t^2) = \frac{1}{1 + exp(-(y_t^2))},$$

$$y_t^2 = w_1^2 z_{t1}^1 + w_2^2 z_{t2}^1 + v^2,$$

$$z_t^1 = (z_{t1}^1, z_{t2}^1) = \hat{\psi}_1(y_t^1) = (\psi_1(y_{t1}^1), \psi_2(y_{t2}^1)),$$

$$= \left(\frac{1}{1 + exp(-(y_{t1}^1))}, \frac{1}{1 + exp(-(y_{t2}^1))} \right),$$

$$y_{t1}^1 = w_1^1 z_{t1}^0 + v_1^1,$$

$$y_{t2}^1 = w_2^1 z_{t2}^0 + v_2^1,$$

$$z_t^0 = (z_{t1}^0, z_{t2}^0) = (k_t, \theta_t).$$

In experiments, we let GAs search not only the network weights, but also

γ_{min} and γ_{max} rather than fixing them ahead. Each experiment starts with a random population, Pop(0). In all simulations, network weights, γ_{min} and γ_{max} are first searched in the interval $[-30, 30]$. If they hit either the lower or the upper bound in half of the runs, then we adjust the search intervals accordingly. No interval adjustments are necessary with one exception. When $\tau = 0.5$ and $\delta = 0$, the GA search for γ_{min} takes place in the interval $[-250, 5]$.

The parameter values used in the simulations are:

$T = 2000, \alpha = 0.33, \delta = \{0,\ 1\}, \rho = 0.95.$

When the exact solution is known, $\delta = \tau = 1$, we adopt $\beta \in \{0.95,\ 0.98\}$ and $\sigma \in \{0.01,\ 0.05\}$. For $\beta = 0.95$, the policy network is trained over the following pairs of (k_0, θ_0):

$\Pi = \{(0.1, 1.49), (0.15, 1.22), (0.25, 1), (0.30, 0.82), (0.34, 0.67)\}$.

For $\beta = 0.98$, on the other hand, the following pairs of (k_0, θ_0) are used for training:

$\Pi = \{(0.008, 7.39), (0.5, 4.48), (1, 2.72), (1.5, 1), (2, 0.61), (2.5, 0.22), (3, 0.37),$
$(3.7, 0.14)\}$.

When the exact solution is not known, $\delta = 0$ and $\tau \in \{0.5,\ 1.5,\ 3.0\}$, we again use $\beta \in \{0.95,\ 0.98\}$ but, only $\sigma = 0.02$. For $\beta = 0.95$, the policy network is trained over the pairs of $(k_0, \theta_0) \in \Pi = K_0 \times \Theta_0$ where

$K_0 = \{10.8, 11.4, 12, 12.6, 13, 13.8, 14, 14.4, 14.8, 15, 15.4, 15.5, 16, 16.2, 16.8,$
$17, 17.4, 18, 18.6, 20.2\},$

$\Theta_0 = \{0.82, 1.0, 1.22\}$.

For $\beta = 0.98$, on the other hand, we train the policy network over the

pairs of $(k_0, \theta_0) \in \Pi = K_0 \times \Theta_0$ where

$K_0 = \{56, 57, 58, 59, 60, 61, 62, 62.5, 63, 63.5, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73\}$,

$\Theta_0 = \{0.82, 1.0, 1.22\}$.

These parameter value choices allow us to make direct comparisons with the results reported in Taylor and Uhlig (1990) and Duffy and McNelis (2001).

The genetic operators were done using the public domain Genesis 5.0 package (Grefenstette, 1990). We compile Genesis 5.0 on a IBM RS/6000 running AIX 5.2. We run the algorithm 10 times and report the best policy functions over all runs. Each run lasts 10000 generations with a *population size* of 50, a *crossover rate* of 0.60, and a *mutation rate* of 0.001. The average time for a specific experiment (10 runs) for one parameter configuration is about 30 minutes. Compared with the existing collocation methods, the computing time may look excessive. However, time efficiency should be judged in the light of the fact that in each run, the GA computes T instantaneous utilities for $|\Pi|$ times for each of the N individuals in each generation. Consequently, the search is relatively slow which is compensated the gain in robustness. Moreover, GAs can be implemented in parallel in order to speed up training.

When the analytical solution is available, we measure the performance of our approximation by the following statistic suggested by Duffy and McNelis

(2001):

$$e(h) = \frac{1}{N_\theta N_k} \sum_\theta \sum_k \left(\frac{\hat{h}(k,\theta) - h(k,\theta)}{h(k,\theta)} \right)^2,$$

where $h(k,\theta)$ is the true policy function, $h(k,\theta) = \alpha\beta\theta k^\alpha$, and $\hat{h}(k,\theta)$ is the approximate policy function, a neural network trained by GA. This accuracy statistic, $e(h)$, is calculated by evaluating each function over a grid of N_k and N_θ values for k and θ.

In particular, 80 equally spaced points between the interval $[-2\sigma, 2\sigma]$ are generated for ε_t. These grid points were then converted into grid points for $\ln(\theta)$ using the long-run relation, $\ln(\theta) = [1/(1-\rho)]\varepsilon$, and for θ by simply taking the exponent of $\ln(\theta)$. To generate grid points for k, the grid points for θ and the long-run relation between k and θ are used, $k = (\alpha\beta\theta)^{1/(1-\alpha)}$. With 80 grid points for k and θ, the error measure, $e(h)$, is thus calculated over 6400 different combinations of k and θ.

The $\log_{10}$ average relative squared error, $e(h)$, 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 1000.

When an analytical solution exists, we also compute the correlation coefficient of the approximate consumption series with the exact consumption series. This statistic is labelled 'corr with exact' in Table 1.

When $\tau \neq 1$, $\delta = 0$, so that no closed-form solution exists, the following four summary statistics are used 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) Euler equation errors as described in Judd (1992). Statistics (1)-(3) are described and were reported in Taylor and Uhlig (1990) for a variety of different solution methods, hence will be only briefly reviewed here.

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

$$\eta_t = \beta c_t^{-\tau}(\alpha\theta_t k_{t-1}^{\alpha-1} + 1 - \delta) - c_{t-1}^{-\tau},$$

$$\hat{a} = (x'x)^{-1}x'\eta,$$

$$\text{DM-Stat} = \hat{a}'(x'x)(x'x\eta^2)^{-1}(x'x)\hat{a},$$

where 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 has an asymptotic Chi-square distribution 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, we use five lags of consumption, c, five lags of the productivity shock, θ, and a constant term as instruments for each sample size of 2000 observations. We use the same set instruments as in Taylor and Uhlig (1990) so that our results can be directly compared with the results reported there. 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 5% level, and critical values [3.05, 24.72] at the 1% level of significance.

The TR^2 statistic ('tr2stat' in the tables) is computed from a regression of the productivity shock, ε, on five lags of consumption, capital, and θ (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 provides another measure of the accuracy of the approximated solution. The following system describes the calculation of the TR^2 statistic.

$$\hat{\varepsilon}_t = x_t \hat{b},$$

$$\hat{b} = (x_t' x_t)^{-1} x_t' \varepsilon_t,$$

$$\text{tr2stat} = T \frac{[\sum(\varepsilon_t - \bar{\varepsilon}_t)(\hat{\varepsilon}_t - \bar{\hat{\varepsilon}}_t)]^2}{\sum(\varepsilon_t - \bar{\varepsilon}_t)^2 \sum(\hat{\varepsilon}_t - \bar{\hat{\varepsilon}}_t)^2},$$

where T again denotes the number of observations in the regression sample, taken to be 2000, and x_t represents the 15×1 vector of lagged values for consumption, capital and θ. 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% significance level are [6.26, 27.49].

The R^2 statistic ('rsqstat' in the tables) is obtained from a regression of the first difference of consumption on lagged consumption and capital, again using a sample of 2000 observations. This test statistic serves as 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.

Judd (1992) proposes the following normalized Euler equation error function as a measure of accuracy:

$$EE(k_t, \theta_t) = 1 - \frac{[\beta E_t(\theta_{t+1} k_{t+1}^{\alpha}, -i_{t+1})^{-\tau}(\alpha \theta_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta)]^{\frac{-1}{\tau}}}{(\theta_t k_t^{\alpha} - i_t)}.$$

Given the current states k_t and θ_t, use of an approximate policy will lead to a suboptimal consumption. The deviation from the truly optimal policy is then measured as the Euler equation error as a fraction of optimal consumption. In other words, this error can be interpreted as the loss in terms of consumption a agents would suffer from by using the approximate solution rather than the true solution. For instance, if $EE(k_t, \theta_t) = 0.01$, then the agent incurs a loss of \$1 for every \$100 in consumption expenditures.[8] In order to ease interpretation, we plot the absolute errors in base 10 logarithms. The plots are drawn for values of capital ranging within the interval $[(1 - \Delta_k)k^*; (1 + \Delta_k)k^*]$ were k^* is the deterministic steady state and $\Delta_k = 0.2$, and value of the technology shock that insures that roughly 95% of the distribution of $ln(\theta_t)$ is covered. The integral involved by the expectation is evaluated using a 20 nodes Gauss-Hermite quadrature.

In addition, for all cases, we present the volatility of the consumption series (denoted as 'con-vol' in the tables), which is simply the standard deviation of the Hodrick-Prescott (HP) filtered series, and the ratio of the vari-

[8]Note that $e(h)$ error metric above is similar to this one.

ance of investment to the variance of the change in consumption (denoted as 'i/c ratio' in the tables).

4.1 When the closed form solution is known

Table 1 reports the best network weights. Table 2 presents the various test statistics for $\beta = \{0.95, 0.98\}$ and $\sigma = \{0.01, 0.05\}$. Also summarized in Table 2 are the values of the same statistics computed using the exact solution.

Table 1: Best network weights when $\tau = \delta = 1$

	β 0.95		β 0.98	
	σ		σ	
	0.01	0.05	0.01	0.05
v_1^1	-1.34897	-0.80156	-1.66178	-0.25415
w_1^1	-8.73900	-5.10264	-5.33724	-1.03617
v_2^1	2.79570	1.11437	-1.62268	0.01955
w_2^1	-2.83480	-1.19257	1.66178	-1.23167
v^2	-0.29326	-0.05865	-2.01369	-0.29326
w_1^2	4.51613	-2.32649	-5.41544	-2.63930
w_2^2	1.66178	-3.77322	2.60020	-6.90127
γ_{max}	-0.05865	2.36559	0.87977	3.85142
γ_{min}	0.68426	-0.05865	-0.05865	-0.01955

The average relative squared error, $e(h)$, reported in Table 2 shows that our method provides highly accurate approximations. This accuracy, however, falls as σ increases. Furthermore, compared with Duffy and McNelis (2001), our algorithm substantially improves approximation accuracy. Note also from Table 2 that the volatility of consumption ('con vol') is almost the same for approximate and exact paths. Moreover, there is a very high correlation between the approximate and the exact consumption paths as

evident from the correlation coefficient reported as 'corr w exact'. The investment/consumption volatility ratio (i/c ratio) is slightly overestimated when σ is low and substantially so when σ increases.[9]

Table 2: Accuracy/diagnostic statistics when $\tau = \delta = 1$*

A. Benchmark Values for Exact Solution

	$\beta = 0.95$			$\beta = 0.98$	
	σ			σ	
	0.01	0.05		0.01	0.05
i/c ratio	0.456638	0.456641	i/c ratio	0.477951	0.477954
con vol	0.006848	0.035386	con vol	0.006853	0.035414

B. Values for Network Approximation

	$\beta = 0.95$			$\beta = 0.98$	
	σ			σ	
	0.01	0.05		0.01	0.05
e(h)	-3.98	-1.37	e(h)	-3.95	-1.22
Duffy-McNelis-NN e(h)	-1.36	-0.26	Duffy-McNelis-NN e(h)	-1.52	-0.24
Duffy-McNelis-PA e(h)	-0.44	-0.38	Duffy-McNelis-PA e(h)	-0.48	-0.39
i/c ratio	0.497581	0.730451	i/c ratio	0.510734	0.793434
con vol	0.006839	0.034683	con vol	0.006834	0.035731
corr w exact	0.999966	0.998840	corr w exact	0.999979	0.997269

*Duffy and McNelis use the parameterized expectations approach. They use both neural network and polynomial parameterizations, which are respectively denoted as 'Duffy-McNelis-NN' and 'Duffy-McNelis-PA' in the table.

4.2 Taylor-Uhlig (1990) Model

In the Taylor-Uhlig version of the model, it is assumed that $\delta = 0$ and $\tau \neq 1$ so that a closed-form solution can not be obtained. Under these parameter

[9]Though not reported in the table, i/c ratio, is overestimated at a higher rate in Duffy and McNelis (2001). Furthermore, the volatility of consumption (con vol) is also overestimated by Duffy and McNelis (2001).

restrictions, the best network weights are displayed in Table 3. Table 4 reports the accuracy and diagnostic test statistics under alternative values of τ. Figure 1 plots the base 10 logarithm of absolute Euler residuals.

Observe from Table 4 that both DM-statistics and 'tr2stat' always lie within the Chi-square accuracy bounds at the 1% significance level for all cases we study. Moreover, the 'rsqstat' values are quite low. Also, the investment/consumption volatility ratios as well as the direct measures of consumption volatility itself are quite similar across cases. Finally, as depicted in Figure 1, our approximation accuracy is reasonably good. Moreover, the experimental results should be judged in the light of the fact that our approach does not make any use of gradient information, is independent of the initial conditions and requires minimal off-line computational effort. Hence, it can complement weighted residual methods especially if the search terrain is highly erratic.

Table 3: Best network weights when $\delta = 0$

	β 0.95 τ			β 0.98 τ		
	0.5	1.5	3.0	0.5	1.5	3.0
v_1^1	23.74976	28.06061	28.06061	-13.41642	-0.91789	0.18377
w_1^1	-1.56891	-1.89150	-1.77419	0.27859	0.07331	0.07331
v_2^1	4.22776	17.86413	17.96188	5.88954	-11.99902	-12.73216
w_2^1	-4.66764	-16.54448	-16.64223	-5.49853	15.22483	16.30010
v^2	7.74682	11.51026	11.85239	19.62366	-2.22385	-0.80645
w_1^2	1.05083	3.08407	2.70283	-11.91105	20.49365	20.87488
w_2^2	-4.32551	-15.32258	-15.71359	-3.93451	-17.22874	-18.15738
γ_{max}	0.36657	0.12219	0.12219	0.61095	-0.41544	-0.46432
γ_{min}	-224.00782	-0.21017	-0.23460	-165.10264	1.51026	4.75073

Table 4: Accuracy/diagnostic statistics when $\delta = 0$

	$\beta = 0.95$			$\beta = 0.98$		
	τ			τ		
	0.5	1.5	3.0	0.5	1.5	3.0
DM-stat	18.503452	20.198500	20.413396	22.595824	15.910241	16.368731
rsqstat	0.038942	0.012267	0.017506	0.004214	0.025213	0.035841
tr2stat	26.017295	14.506097	14.357957	19.595485	12.666643	12.385571
i/c ratio	2.604381	2.033353	2.016889	3.888067	2.477865	2.697584
con vol	0.036475	0.035060	0.035415	0.033837	0.042774	0.041537

Table 5 summarizes the accuracy and diagnostic test statistics from our approximation (Genetic NNs), and compare these statistics with those obtained from a subset of the other solution methods presented in Taylor and Uhlig (1990) and Duffy and McNelis (2001). In particular, we compare our solution method with the log-linear quadratic (log-LQ) and linear quadratic LQ solution methods of Christiano (1990) and McGrattan (1990), the backsolving methods of Ingram (1990) and Sims (1990), the parameterized expectations approach of Den Haan and Marcet (1990), the parameterized expectation approach using neural network and polynomial approximations (Duffy/McNelis-NN and Duffy/McNelis-PA) of Duffy and Mcnelis (2001) and the quadrature method of Tauchen (1990). Inspection of Table 5 indicates that our approach compares favorably on many dimensions with these alternative and more commonly used methods. In particular, our solutions are the only ones that are consistent with the random walk behavior of consumption, produce reasonable consumption-investment volatility ratios and also pass tests of the martingale difference property of both the residuals of the Euler equation and the productivity shocks in all cases we consider.

Table 5: Comparison with alternative methods with $\delta = 0$

	$\beta = 0.95,\ \tau = 0.5$				$\beta = 0.95,\ \tau = 1.5$				$\beta = 0.95,\ \tau = 3.0$			
	DM-stat	TR^2 stat	R^2	i/c ratio	DM-stat	TR^2 stat	R^2	i/c ratio	DM-stat	TR^2 stat	R^2	i/c ratio
Genetic NNs	18.50	26.02	0.04	2.60	20.20	14.51	0.01	2.03	20.41	14.36	0.02	2.02
Duffy/McNelis-NN	35.46	13.69	0.02	9.75	7.98	11.89	0.03	9.92	2.34	11.46	0.05	13.28
Duffy/McNelis-PA	34.38	12.63	0.07	5.77	36.12	14.09	0.07	3.24	25.96	15.42	0.04	1.94
Christiano-Loq LQ	17.00	10.00	0.43	29.00	10.00	10.00	0.05	11.00	18.00	19.00	0.02	8.00
Ingram	10.00	17.00	0.44	30.00	11.00	165.00	0.06	12.00	12.00	394.00	0.03	20.00
Den Haan/Marcet	18.00	15.00	0.42	30.00	18.00	14.00	0.06	13.00	12.00	13.00	0.03	10.00
McGratten	96.00	19.00	0.34	24.00	22.00	19.00	0.04	9.00	17.00	19.00	0.02	7.00
Sims	12.00	24.00	0.44	31.00	12.00	24.00	0.07	13.00	12.00	22.00	0.04	11.00
Tauchen	704.00	11.00	0.50	3.00	558.00	9.00	0.38	2.00	502.00	14.00	0.33	2.00
	$\beta = 0.98,\ \tau = 0.5$				$\beta = 0.98,\ \tau = 1.5$				$\beta = 0.98,\ \tau = 3.0$			
	DM-stat	TR^2 stat	R^2	i/c ratio	DM-stat	TR^2 stat	R^2	i/c ratio	DM-stat	TR^2 stat	R^2	i/c ratio
Genetic NNs	22.60	19.60	0.00	3.89	15.91	12.67	0.03	2.48	16.37	12.39	0.04	2.70
Duffy/McNelis-NN	22.44	12.16	0.01	6.53	13.66	8.34	0.02	8.68	8.93	14.08	0.02	8.56
Duffy/McNelis-PA	30.76	9.77	0.35	5.74	21.23	11.96	0.33	46.55	31.00	14.27	0.04	3.36
Christiano-Loq LQ	28.00	20.00	0.24	132.00	16.00	25.00	0.03	59.00	12.00	16.00	0.01	45.00
Ingram	8.00	15.00	0.33	162.00	11.00	203.00	0.04	66.00	12.00	381.00	0.02	98.00
Den Haan/Marcet	30.00	15.00	0.35	178.00	7.00	14.00	0.04	78.00	9.00	14.00	0.02	74.00
McGratten	62.00	19.00	0.21	112.00	26.00	17.00	0.02	44.00	21.00	16.00	0.01	38.00
Sims	11.00	19.00	0.36	171.00	12.00	16.00	0.04	66.00	10.00	14.00	0.02	59.00
Tauchen	322.00	16.00	0.34	2.00	234.00	13.00	0.27	2.00	215.00	10.00	0.27	2.00

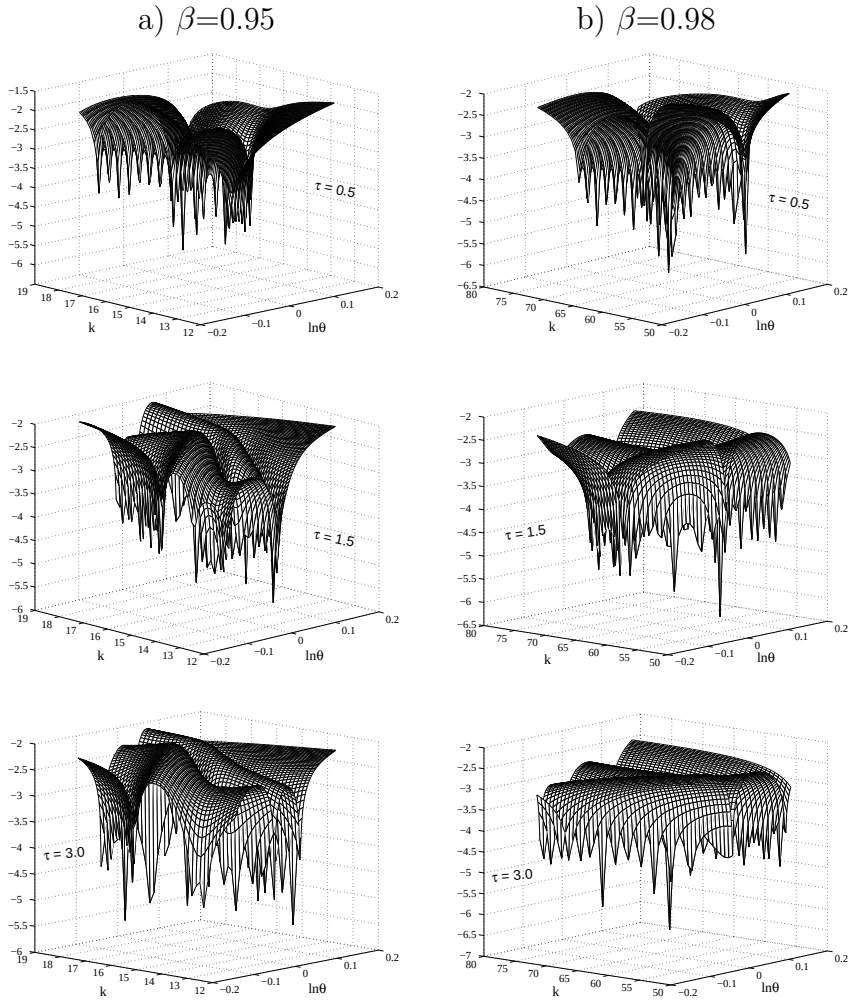

Figure 1: $\text{Log}_{10}|\text{Euler Residuals}|$

5 Conclusion

This paper has shown that a direct numerical optimization approach wherein investment policy is parameterized by a neural network and trained by a genetic algorithm can be a useful alternative to existing numerical solution

methods to the stochastic growth model. For the special case of the one-sector stochastic growth model where an exact solution is available, our solution method provides highly accurate approximations. When analytic solutions are not available, our approximation accuracy as measured by the Euler equation error is reasonably good. Furthermore, in contrast with the numerical methods reported in Taylor and Uhlig (1990) and Duffy and McNelis (2001), only our genetically evolved neural networks produce solutions with reasonable consumption-investment volatility ratios that pass tests of the martingale difference property of both the Euler equation residuals and the productivity shocks, and are also consistent with the random walk behavior of consumption.

The algorithm is on-line and general purpose. It can easily be extended to models with higher degrees of non-linearity, larger state spaces and possible discontinuities. Both neural networks and genetic algorithms are parallel paradigms for multiple search with modest memory requirements. Therefore, a search with genetic neural networks is robust but, a bit time consuming. This, however, can be ameliorated by running the genetic neural networks on multiple processors synchronously, thanks to their amenability to explicit parallelism.

References

[1] Alemdar, N.M. and S. Özyıldırım, A genetic game of trade, growth and externalities. *Journal of Economic Dynamics and Control* **22**, 811-32 (1998).

[2] Alemdar, N.M. and S. Özyıldırım, Learning the Optimum as a Nash Equilibrium. *Journal of Economic Dynamics and Control* **24**, 483-499 (2000).

[3] Branke, J., Evolutionary Algorithms for Neural Network Design and Training. *Technical Report*, **No. 322**, University of Karlsruhe, Institute AIFB (1995).

[4] Brock, W.A. and L. Mirman, Optimal economic growth and uncertainty: the discounted case. *Journal of Economic Theory* **4**, 479-513 (1972).

[5] Christiano, L.J., Solving the Stochastic Growth Model by Linear quadratic Approximation and by Value Function Iteration. *Journal of Business and Economic Statistics* **8**, 99-113 (1990).

[6] Christiano, L.J. and J.D.M Fisher, Algorithms for solving dynamic models with occasionally binding constraints. *Journal of Economic Dynamics and Control* **24(8)**, 1179-1232 (2000).

[7] Cooley, T.F. (ed.), *Frontiers of Business Cycle Research.* Princeton University Press, Princeton NJ (1995).

[8] den Hann, W.J. and A. Marcet, Solving the stochastic growth model by parameterizing expectations. *Journal of Business and Economic Statistics* **8**, 31-34 (1990).

[9] den Haan,W.J. and A. Marcet, Accuracy in simulations.*Review of Economic Studies* **61**, 3-17 (1994).

[10] Duffy, J. and P.D. McNelis, Approximating and simulating the stochastic growth model: Parameterized expectations, neural networks, and the genetic algorithm. *Journal of Economic Dynamics and Control* **25**, 1273-1303 (2001).

[11] Eaton, J., Fiscal Policy, Inflation, and the Accumulation of Risky Capital. *Review of Economic Studies* **48**, 435-445 (1981).

[12] Fair, R. C. and J. B. Taylor, Solution and maximum likelihood estimation of dynamic nonlinear rational expectations models. *Econometrica* **51**, 1169-1186 (1993).

[13] Fletcher, C.A.J., *Computational Galerkin Techniques*. New York: Springer-Verlag (1984).

[14] Gagnon, J.E., Solving the stochastic growth model by deterministic extended path. *Journal of Business and Economic Statistics* **8**, 3536 (1990).

[15] Grefenstette, J.J., A User's Guide to GENESIS Version 5.0. *Manuscript* (1990).

[16] Hecht-Nielsen, R., *Neurocomputing*. Massachusetts, Addison-Wesley Publishing Company (1990).

[17] Heer, B., and A. Maussner, *Dynamic General Equilibrium Models: Computation and Applications*. Springer (2005).

[18] Hertz, J., Krogh, A., A. G. Palmer, *Introduction to the Theory of Neural Computation*. Massachusetts, Addison-Wesley Publishing Company (1991).

[19] Holland, J.H., *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI (1975).

[20] Hornik, K., Stinchcombe, M. and H. White, Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, **Vol. 2**, 359-366. (1989)

[21] Ingram, B.F., Solving the Stochastic Growth Model by Backsolving with an Expanded Shock Space. *Journal of Business and Economic Statistics* **8**, 37-38 (1990).

[22] Jones, C.I., Time Series Tests of Endogenous Growth Models. *Quarterly Journal of Economics* **110**, 395-527 (1995).

[23] Judd, K.L., Projection methods for aggregate growth models. *Journal of Economic Theory*, **58**, 410-452 (1992).

[24] Judd, K.L., *Numerical Methods in Economics*. MIT Press, Cambridge MA (1998).

[25] Kim, J. and S.H. Kim, Spurious Welfare Reversals in International Business Cycle Models. *Journal of International Economics* **60**, 471-500 (2003).

[26] King, R.G., C.L. Plosser, and S.T. Rebelo, Production, Growth, and Business Cycles I: The Basic Neoclassical Model. *Journal of Monetary Economics* **21**, 191-232 (1988).

[27] McGrattan, E.R., Solving the Stochastic Growth Model by Linear-Quadratic Approximation. *Journal of Business and Economic Statistics* **8**, 41-44 (1990).

[28] McGrattan, E.R., Application of Weighted Residual Methods to Dynamic Economic Models. In Marimon, R., and A. Scott (Eds.), *Computational Methods for the Study of Dynamic Economies*, 114-142 (1999).

[29] Miranda, J.M. and P.L. Fackler, *Applied Computational Economics and Finance*. MIT Press, Cambridge MA (2002).

[30] Mirrlees, J.A., Optimum Accumulation under Uncertainty. *unpublished manuscript*, (1965).

[31] Mühlenbein, H., Darwin's continent cycle theory and its simulation by the prisoner's dilemma. In Verela, F.J., and P. Bourgine (Eds.), *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 236-244 (1992).

[32] Narenda, K. S. and K. Parthasarthy, Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transaction on Neural Networks* **Vol.1, No.1**, 4-27 (1990).

[33] Romer, P. M., Increasing Returns and Long-run Growth. *Journal of Political Economy* **94**, 1002-1037 (1986).

[34] Sargent, T.J., *Dynamic Macroeconomic Theory.* Harvard University Press, Cambridge MA, USA (1987).

[35] Schmitt-Grohé, S. and M. Uribe, Solving Dynamic General Equilibrium Models using a Second-order Approximation to the Policy function. *Journal of Economic Dynamics and Control* **28**, 755-775 (2004).

[36] Sims, C. A., Solving the Stochastic Growth Model by Backsolving with a Particular Nonlinear Form for the Decision Rule. *Journal of Business and Economic Statistics* **8**, 45-47 (1990).

[37] Sirakaya, S. and N. M. Alemdar, Genetic Neural Networks to Approximate Feedback Nash Equilibria in Dynamic Games. *Computers and Mathematics with Applications* **Vol 46/10-11**, 1493-1509 (2003).

[38] Taylor, J.B., H. Uhlig, Solving nonlinear stochastic growth models: a comparison of alternative solution methods. *Journal of Business and Economic Statistics* **8**, 1-17 (1990).

[39] Tauchen, G., Solving the stochastic growth model by using quadrature methods and value-function iterations. *Journal of Business and Economic Statistics* **8**, 4951 (1990).

[40] Turnovsky, S.J., *Methods of Macroeconomic Dynamics*. MIT Press, Cambridge MA (2000).