Approximating and Simulating the Stochastic Growth Model:

Parameterized Expectations, Neural Networks, and the Genetic Algorithm

John Duffy
Department of Economics
University of Pittsburgh
Email: jduffy+@pitt.edu

Paul D. McNelis
Department of Economics
Georgetown University
Email: mcnelisp@gunet.georgetown.edu

October 1997

Abstract

This paper compares alternative methods for approximating and solving the stochastic growth model with parameterized expectations. We compare polynomial and neural network specifications for expectations, and we employ both genetic algorithm and gradient-descent methods for solving the alternative models of parameterized expectations.

Many of the statistics generated by the neural network specification in combination with the genetic algorithm and gradient descent optimization methods approach the statistics generated by the exact solution with risk aversion coefficients close to unity and full depreciation of the capital stock. For the alternative specification, with no depreciation of capital, the neural network results approach those generated by computationally-intense methods. Our results suggest that the neural network specification and genetic algorithm solution methods should at least complement parameterized expectation solutions based on polynomial approximation and pure gradient-descent optimization.

I. Introduction

This paper compares alternative methods for approximating and solving the stochastic growth model with parameterized expectations. In particular, we distinguish between polynomial and neural network specifications for expectations, and between gradient-descent and genetic-algorithm methods for solving the alternative models of parameterized expectations.

Parameterized expectation methods require specification of the mechanism by which expectations are formed as well as a method for finding the parameters of this expectations mechanism. Most approaches to parameterized expectations rely on a polynomial approximation for the expectations mechanism and a gradient-descent-based method for finding the appropriate parameters values of the expectations mechanism.

While other computationally more intensive methods are also used, such as the quadrature-based methods described by Tauchen and Hussey (1991), there has been little application of artificial neural networks or genetic algorithms to the stochastic growth model.[1]

Parameterized expectation methods based on polynomial specifications and gradient-descent methods 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.

---

[1] Beaumont and Bradshaw (1995) are the only other researchers we are aware of who have used genetic algorithms to find solutions to optimal growth models. These authors use a distributed parallel genetic algorithm running on a cluster of RISC workstations to search for the coefficients to a Chebyshev polynomial approximation of the policy function of the optimal stochastic growth model. By contrast, we consider a neural network approximation to the policy function. Furthermore, our approach is easily implemented on a single PC or workstation. Other related approaches include Chyi's (1997) use of neural network methods to approximate policy functions and Semmler and Gong (1997)'s use of simulated annealing to estimate stochastic growth models.

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

A genetic-algorithm-based search for optimal parameter values also differs markedly from the gradient descent methods that are typically used. The genetic-algorithm search is a population-based, evolutionary search process that begins with a very large set of initial candidate parameter vectors. The objective of this stochastic search process is to find the parameter values (solutions) that maximize a given objective function, e.g. utility. 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 can greatly enhance the performance of "locally efficient gradient-type algorithms" by providing the "needed global efficiency" [Dorsey and Mayer (1995): p. 565]. [2] 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 the neural network and genetic algorithm approaches may give more accurate results than the more commonly implemented parameterized expectations methods based on polynomial approximation and gradient-

[2] Dorsey and Mayer (1995) analyzed the performance of the GA for static non-linear econometric estimation. Our work is thus an extension of their work. We analyze the performance of the GA for the solution of dynamic non-linear optimization models, rather than estimation.

descent optimization. While computational time considerations may prevent the genetic algorithm from taking the place of faster methods, 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 approximation, as well as other solutions, with results they obtain using neural network approximations and genetic algorithms.

This paper is inspired by the recent work of Den Haan and Marcet (1994), who compared the accuracy of the linear quadratic (LQ), the log LQ and parameterized expectations (PE) methods for solving the standard stochastic optimal growth model (see Taylor and Uhlig (1990)) and for the real business cycle model with money due to Cooley and Hansen (1989). In this paper, Den Haan and Marcet propose a test for accuracy based on the sum of squares of the first derivative of the utility function, with solutions based on the LQ, log LQ and PE methods. In their study, the parameterized expectation was implemented with a second-order polynomial approximation to the policy function. They found that the LQ methods broke down under a relatively high variance for the disturbance term.

We concentrate on the parameterized expectation framework because of its greater accuracy, and because of its flexibility. Other methods which linearize around a steady state are faster, to be sure, but there may instances when the assumption of a steady state may be too restrictive.

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 in Taylor and Uhlig (1990), Den Haan and Marcet (1994) and many others. We also present a brief review of the parameterized expectations approach. Section III is a summary of our simulation results. The last section concludes.

II. A Neural Network/Genetic Algorithm For Solving the Stochastic Growth Model

Consider the following one-sector stochastic growth model:

$$
\begin{aligned}
&Maximize: \\
&\sum_{i=0}^{\infty} \beta^{i} U(c_{t+i}) = \sum_{i=0}^{\infty} \beta^{i} \frac{c_{t+i}^{1-\tau}}{1-\tau} \\
&s.t. \\
&c_{t+i} + k_{t+i} - k_{t+i-1} = A\theta_{t+i} k_{t+i-1}^{\alpha} - \mu k_{t+i} \\
&\ln(\theta_{t+i+1}) = \rho \ln(\theta_{t}) + \varepsilon_{t+1} \\
&\varepsilon_{t} \sim N(O, \sigma^{2}) \\
&c_{t+i} > 0, k_{t+i} > 0, \forall i
\end{aligned}
\qquad (1)
$$

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, μ is the constant rate at which the capital stock depreciates from t-1 to t, and θ_t denotes the time t stochastic shock to technology that evolves according to an autoregressive process with parameter $|\rho| < 1$, and an error process $\varepsilon_t \sim N(0,\sigma^2)$. The solution to this optimization problem yields a policy function:

$$c_t = h(k_t)$$

and a law of motion for the stock of capital:

$$k_{t+1} = g(k_t) = A\theta_t k_t^{\alpha} + (1-\mu)k_t - h(k_t).$$

It can be shown that equation system (1) has a closed form analytical solution only when $\tau=\mu=1$ (i.e. only in the case of logarithmic preferences and full depreciation of the capital stock in every period). In this special case, the optimal policy function and path for the capital stock are given by the following system:

$$
\begin{aligned}
c_t &= (1-\alpha\beta)\theta_t A k_t \,, \\
k_{t+1} &= \alpha\beta\theta_t A k_t \,.
\end{aligned}
\qquad (1')
$$

For all other cases, numerical approximation methods are needed to solve for the optimal policy function and capital accumulation path for given stochastic shocks.

The method of parameterized expectations, due to Den Haan and Marcet (1990, 1994) takes seriously the Euler equation for the stochastic model:

$$c_t^{-\tau} = \beta E_t[c_{t+1}^{-\tau}\theta_{t+1}k_t^{\alpha-1}\alpha + 1 - \mu]. \qquad (2)$$

The method proceeds in the following three steps: first, substitute the conditional expectation on the right-hand side of equation (2) by a parameterized function $\psi(\gamma; k_{t-1},\theta_t)$, in order to obtain:

$$c_t^{\tau} = \beta\psi(\gamma; k_{t-1},\theta_t). \qquad (3)$$

Secondly, create a long series of {c,k} with equation (2) and (3); third, run a non-linear regression:

$$\lambda_{t+1} = \psi(\gamma; k_{t-1},\theta_t) \qquad (4)$$

which iterates until γ coincides with the result of the non-linear regression, and thus minimizes the differences between the actual and expected values of λ_{t+1}.

A common way to parameterize equation (3) is the following polynomial logarithmic expansion:

$$c_t^{\tau} = \gamma_0 + \gamma_1 \ln(k_{t-1}) + \gamma_2 \ln(\theta_{t-1}) + \gamma_3 \ln(k_{t-1})^2 + \gamma_4 \ln(k_{t-1})\ln(\theta_t) + \gamma_5 \ln(\theta_t)^2 \qquad (5)$$

The neural network alternative involves the same number of parameters, but uses a logsigmoid expansion term rather than a polynomial expansion:

$$\begin{aligned} c_t^{\tau} &= \tilde{\gamma}_0 + \tilde{\gamma}_1 \ln(k_{t-1}) + \tilde{\gamma}_2 \ln(\theta_{t-1}) + \tilde{\gamma}_3\Omega_t \\ \omega_t &= \tilde{\gamma}_4 \ln(k_{t-1}) + \tilde{\gamma}_5 \ln(\theta_{t-1}) \\ \Omega_t &= \frac{1}{1+e^{\omega_t}} \end{aligned} \qquad (6)$$

The logsigmoid function is widely used in neural network approximation methods.[3] As Sargent (1993) has pointed out, this function is more accurate and parsimonious than polynomial methods. In neural network estimation, the coefficient set $[\tilde{\gamma}]$ is obtained through the backpropagation method, an interative 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's parameterized expectation method operates in a similar manner on the coefficient of the polynomial expansion.

An alternative solution technique is to use a *genetic algorithm* . The genetic algorithm attempts to solve optimization problems, whether static or dynamic, using a large population of candidate solutions that are subjected to some type of 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.[4]

The genetic algorithm approach is different from the gradient descent method in that it begins with not just one, but a large set of randomly generated parameter vectors known as "bit strings" because they are typically encoded using a binary alphabet. In our application of the genetic algorithm, the elements of each string consist of a vector of coefficients that are to be 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 consist of five different values -- one for each

[3] Equation (7) would be classified as a single hidden-layer logsigmoid neural network with one neuron. See Bishop (1995) for an up-to-date treatment of neural network methods.

[4] Mitchell (1996) provides a good introduction to genetic algorithms.

of the five coefficients in either the polynomial or neural network specifications, equations (5) or (6). Thus each string represents a different candidate polynomial or neural network approximation for the conditional expectation. Moreover, the elements of our strings are real ,base 10 numbers, not the binary encodings that are often encountered in genetic algorithm applications.[5] The size of this population of parameter vectors is determined by the researcher. In our application, we consider populations of size 40.

The genetic algorithm proceeds in a 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 value of each string is determined using a given objective function, e.g. a utility function -- this value serves as the bitstring's *fitness* value. In our application, the fitness function is the sum of squared errors. The fitness values of the selected pair of strings are compared. The string with the highest fitness value is retained for "breeding purposes." This simple tournament selection process is intended to mimic the evolutionary notion of survival of the fittest.[6] 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 preceeding generation of strings.

The strings in this breeding population are then randomly paired and recombined via a process known as crossover. The crossover operator is applied to each of these pairs of strings with probability $p^c > 0$. If a pair of strings is subjected to crossover, a random integer is drawn from the range 0 to the length of the string, which in our application is

[5] Real number encodings are increasingly being used in genetic algorithm applications as they avoid the time consuming conversion into and out of the base 10 number system. In many instances, real value encodings have been shown to outperform bitstring encodings. See, e.g. Davis (1991).

[6] 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).

fixed at 5. The pair of strings is cut using the chosen integer value and the elements of the string to the right of this cut-point are then swapped. The two recombined strings can be viewed as the "offspring" or "children" of the original pair of "parent" strings. These two recombined strings are then subjected to some mutation. The mutation operation is applied to each element in these two recombined strings with some small probability $p^M > 0$. If mutation is to be performed on an element in the string, a random number is drawn from a standard normal distribution. This random number is then added to the element of the string.

Finally a selection tournament takes place 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 will take their place as members of the next generation of candidate strings. The other two strings are discarded. This last step, known as the election operator (see Arifovic (1994)) 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.[7]

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, others are new children.

After a given number of generations, the best coefficient set (the one with the highest 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.[8]

---

[7] The election operator also serves to endogenously contain the destructive effects of the mutation operator as one gets closer to an optimal solution.

[8] A more formal mathematical representation of the genetic algorithm appears in Appendix I.

While it is clear from the process described above that only above average coefficient strings will survive successive populations, one can insure that the very best string will survive from one generation to the next by applying elitism.[9] A given coefficient string $\tilde{\gamma}$(I*,k) from generation I*, ranking first in this generation, may not be chosen for crossover to generation I*+1. Under elitism, if the fitness criterion applied to all the members of generation I*+1 does not dominate $\tilde{\gamma}$ (I*,k), then $\tilde{\gamma}$ (I*,k) will be allowed to survive to generation I*+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. However, the mutation operator cuts with a two-edged sword. Applying mutation too frequently will result in destroying highly fit strings in the population, and thus may slow down or impede convergence to a solution. Back (1993) has conjectured that the optimal mutation rate should be (1/n), where n is the number of coefficients or elements in a bit string or coefficient vector; in our case n=5.

In our analysis, we set the mutation rate at 1/5 =.2. The crossover probability is .9, the population of strings in each generation is 40. To minimize on computing time, the maximum number of generations for finding the “fittest” string is 50. The resulting string is the product of the 'pure' genetic algorithm optimization. In some simulations, we continued the search for the best coefficient vector even further, taking the fittest string of coefficients as determined by 50 iterations (generations) of the genetic algorithm and using this string to initialize a standard gradient descent algorithm. The gradient descent algorithm was then allowed to iterate for 500 iterations.

[9] Arifovic (1994) provides an economic application of the genetic algorithm that illustrates the importance of elitism.

This paper examines the outcomes of the various different methods, for given parameter values of the stochastic model:

$$\alpha = .33, \mu = \{1,0\}, \rho = .95$$

and with varying values of β (.95, .98), σ (.02 , .1) and τ (.5, 1.5,3). The results appear in the following section.[10]

III. Simulation Results

In Table I, we discuss the simulation results for the case of τ=1, under the exact solution method, with full depreciation, μ=1. These statistics serve as "benchmark" values, from which we can compare the statistics generated for alternative values of τ. In Tables II through VI we compare the paths for the cases of τ=.5 , 1.5, and 3, for the polynomial and neural network specifications, under pure genetic algorithm estimation, and under the combined ga/gradient descent optimization. Finally, in Tables VII and VIII we compare the results of our polynomial and neural network with mixed genetic algorithm and gradient-descent approaches, with results reported by Taylor and Uhlig (1990) for alternative solution methods, under the same configurations of parameter values, but with zero depreciation, μ = 0.

For all of the solution methods, we compute the volatility of the consumption series, defined as the variance of the Hodrick-Prescott (HP) filtered series, as well the following four summary statistics: *(1)* the Den-Haan-Marcet statistic, *(2)* the TR^2 statistic, *(3)* the R^2 test, and *(4)* the ratio of the variance of investment to the variance of the first difference of consumption. All of these statistics are described and reported in Taylor and Uhlig (1990) and more recently in Chiyi (1997).

The Den-Haan-Marcet m statistic is computed in the following way:

[10] Following Taylor and Uhlig (1991), we do not report the results for the case of $\sigma = 1$ and $\tau = 3$.

$$\eta_t = \beta c_t^{-1}(\alpha\theta_t k_{t-1}^{\alpha-1} + 1 - \mu)c_{t-1}^{\tau} - 1$$
$$\hat{a} = (x'x)^{-1}x'\eta \qquad (7)$$
$$m = \hat{a}'(x'x)(x'x\eta^2)^{-1}(x'x)\hat{a}$$

where x is a matrix of instrumental variables, lagged c and θ. The statistic m is distributed as a Chi-square variable with degrees of freedom equal to number of instruments, under null-hypothesis of accurate approximation to optimal path.

The TR^2 is computed from a regression of the productivity shock ε on five lags of consumption, capital, and θ, The following system describes the calculation of this statistic:

$$\hat{\varepsilon}_t = x_t\hat{b}$$
$$\hat{b} = (x_t^{'}x_t)^{-1}x_t^{'}\varepsilon_t \qquad (8)$$
$$TR^2 = T\frac{[\sum(\varepsilon_t - \bar{\varepsilon}_t)(\hat{\varepsilon}_t - \bar{\hat{\varepsilon}}_t)]^2}{\sum(\varepsilon_t - \bar{\varepsilon}_t)^2\sum(\hat{\varepsilon}_t - \bar{\hat{\varepsilon}}_t)^2}$$

The R^2 is simply a test of the random walk hypothesis of consumption. It comes from a regression of the first difference of consumption on lagged consumption and capital.

The ratio of the variance of investment to the variance of the first difference of consumption is a frequently discussed feature of economic fluctuations.

A. Simulation Results with τ=1, μ=1

Table I presents the various test statistics for the case of the known closed form solution where τ=1, and there is full depreciation.

The volatility measure increases by a factor of eight, as σ increases by a factor of five. Under a log-linear system, one expects the Den-Haan-Marcet, the TR^2, and the R^2 statistics to be relatively small, and some well within the confidence limits of their theoretical distributions [Uhlig and Taylor (1991): p. 13].

The reason we report these statistics is to compare differences for changes in τ, and thus to assess how much of a difference the abandonment of a log-linear framework makes for key statistical properties of this model.

B. Simulation of Non-linear Models with μ=1

Tables II through VI contain, in turn, the measures for consumption volatility, the Den-Haan/Marcet, TR^2, and R^2 statistics, and the investment/consumption volatility ratio. Each table has results for both the polynomial and neural network approximations, with either the ga or the combined ga/gradient descent estimation.

B.1. The consumption volatility measures

In Table II, the pure genetic algorithm, and the combined ga/gradient descent algorithm give values for consumption volatility that in most cases range between .01 and .15 for τ =.5 and τ=1.5, under σ =.02 and .1, and β=.95 and .98. This range is not unreasonable, since the exact volatility measures for the case of τ=1 fall between .01 and .08, for the same values of σ and β. Since τ=.5 or 1.5 are not very far from unity, one may conjecture that the consumption volatility measures under these assumptions should not be "very far" from those generated when τ=1.

However, there are several striking "outliers" in the polynomial approximation model. With σ=.1, for β=.95 and β=.98, the volatility under the pure ga estimation is several orders of magnitude away from the more reasonable values of .04 and .05 that are obtained when the gradient descent method is used in combination with the ga.

Table II also shows that the neural network approximation model does not produce outliers of such magnitude. While the pure genetic algorithm estimation is often off the mark, the combined genetic algorithm/gradient descent estimation results in volatility measures that seem reasonable and in all cases are greater than the values for the pure ga estimation.

For the case of τ=3, of course, we stray into unknown territory, since we have no expectation of what a "reasonable" estimate of consumption volatility should be. For the polynomial approximation model, under σ=.02, the value of the consumption volatility measure under the ga/gradient descent estimation "jumps" from .16 to .63 as β increases from .95 to .98. In the neural network approximation model, by contrast, the corresponding measures increase only slightly, from .70 to .72.

Overall, the neural network model shows that consumption volatility always increases as β increases, whatever the value of τ. The polynomial model is not so consistent. While there is no reason to expect volatility to be positively or negatively related to β on a priori grounds, since the system is non-linear, the broad consistency of the network specification, indicating uniformly small positive effects of β on consumption volatility, is less troublesome.

B.2 Den Haan-Marcet Statistics

The Den Haan-Marcet statistics appear in Table III. Under the null hypothesis of a martingale difference for η in equation (7), this statistic has approximately a $\chi^2(11)$ distribution, with lower and upper critical values at 3.82 and 21.92 for a significance level of 2.5%, with μ=0, no depreciation. Under μ=1, however, there is no reason to expect a martingale difference. We see this confirmed both in Table I and in Table III: neither the exact solution nor the approximate solutions show any evidence of a martingale difference for η.

The exact solution estimates in Table I show that the Den-Haan-Marcet statistics decrease with an increase in σ for a given β, and rise with β for a given σ. In Table III there is no such pattern, neither for the polynomial nor neural network approaches, under the pure genetic algorithm and the combined genetic algorithm gradient descent approach, even when we are "close" to the log-linear model, with τ=.5 or τ=1.5.

B.3 The TR^2 Statistics

The TR^2 statistic is less problematic than the Den Haan-Marcet statistic. Since the productively shock ε was generated randomly, the test statistic is $\chi^2(16)$ by construction. The range given by the critical values is between 6.9 and 28.8.

In the case of the exact solution, with τ=1, we see in Table I that this statistic rises with σ and for σ=.1, falls with an increase in β.

Table IV shows that neither the polynomial nor neural network approximations replicate this behavior exactly. However, the neural network model, under the genetic

algorithm as well as the genetic algorithm/gradient descent estimation shows a negative relation between the TR^2 statistic and β, for σ=.1.

B.4 The R^2 Statistics

The R^2 statistic simply tests the random walk hypothesis of consumption with an R^2 close to zero. Table I shows that the exact solution has this characteristic. The statistic is invariant with respect to β and σ.

In Table V, there is no clear pattern with this statistic under both the polynomial and network models, with the genetic algorithm and the genetic algorithm with gradient descent. The statistic ranges from relatively low values to .99. It is fair to say that the evidence does not favor random walk behavior in consumption, even for relatively small departures from log-linearity.

B.5 The Investment/Consumption Volatility Ratios

Under the exact solution, Table I shows that the investment/consumption volatility ratio falls with an increase in σ, given β, while it rises with β, given σ.

Table VI shows there is no clear pattern in either approach. However, for the case of β=.95, with σ=.02, the neural network model with the combined g-a and gradient descent shows that this ratio steadily falls as τ, the coefficient of relative risk aversion rises. This is reasonable behavior: with greater aversion to risk, there would be less willingness to invest, and more willingness to consume in the present, and thus a lower value for this ratio. However, the neural network detects this pattern in only one case. In

other instances, both the neural network and the polynomial models generate outliers several orders of magnitude away from the exact solution.

The behavior of this statistic indicates the difficulties estimating volatility with non-linear approximation, even for "small" deviations from the pure log-linear model. Unfortunately, as we show in the next section, this difficulty is shared by practically all of the other approximation methods.

C. Simulation Results with $\mu = 0$

Tables VII and VIII contain the Den-Haan and Marcet, the TR^2, the R^2, and the investment/consumption volatility statistics for this model, with no depreciation, with the same configurations for β, σ, and τ. We present the statistics for the neural network (NN) and polynomial approximations (PA) with the combined genetic algorithm/gradient descent solutions only. We also present the corresponding statistics based on alternative solution methods, reported in Taylor and Uhlig (1990), for the same parameter sets.

The Marcet results are based, of course, on parameterized expectations with a polynomial approximation. Table VI shows that in three cases, in Panel A, Panel B and Panel C, the Den-Haan-Marcet statistics calculated with the combined genetic algorithm/gradient descent technique fall relatively close to the values reported by Marcet for the same statistic. However, in Panels E and F the neural network generates Den Haan-Marcet statistics quite close to those reported by Marcet. In general, though, the Den-Haan-Marcet statistics generated by the combined genetic algorithm and gradient descent estimation fall closer to those reported by Tauchen, who also made use of a computationally intensive method.

The TR^2 statistics fall within the ranges given by the other methods, except in Panel A of Table III, where the "na" simply means that this statistic diverged to a very large number.

Ideally, one would like to obtain low R^2 statistics. But one sees that the values for this statistic jumps around for our methods as they do for the other methods. In several cases, Panels B and D of Table VII and Panel B of Table VIII show that R^2 statistics for the neural network model are close to the values reported by Tauchen. In other instances they are quite small.

The investment/consumption volatility ratio also varies a great deal, across models for the same parameter configuration, and across parameter values within the same model. In Panels B,D E and F of Table VII and Panels B and D of Table VIII this value became so large, several orders of magnitude beyond the range of the data, that we report it simply as "na" in the Table.

Ideally we would like to report values for this ratio greater than one, but not absurdly greater than one. The exact solution gives a ratio between 2 and 2.5, under the case of full depreciation. With the exception of the results reported by Tauchen, the neural network produces ratios close to this range in two cases, Panels A and C of Table VIII, with $\tau = .5$ and $\sigma = .1$. The polynomial model produces values in this range in only one case, Panel C of Table VIII.

The desired result would be a panel with Den-Haan and Marcet values between 3.8 and 22, TR^2 statistics between 6.9 and 28.8, R^2 statistics close to zero, and of course investment/consumption volatility ratios between 2 and 3. None of the methods produce such a result.

We believe that the "best" result in Tables VII and VIII is Panel C of Table VIII, under the neural network specification. While the result is outside the acceptable range of the Den-Haan-Marcet statistic, it has a lower R^2 than the Marcet specification, an acceptable TR^2 statistic, and an investment/consumption volatility ratio of 2.21. Only the Tauchen method, another computationally intensive approach, comes close to these results.

The neural network does not deliver such "desirable" results under all of the parameter configurations, to be sure. Coupled with the genetic algorithm the polynomial model also does quite well in Panel C of Table VIII, beating out the Marcet specification in terms of the investment/consumption volatility ratio.

The message of these two tables is that the genetic algorithm, coupled with the polynomial or neural network approximation to parameterized expectations, should at least compliment the use of pure gradient-descent and other approaches for solving the stochastic growth model.

IV. Conclusion

This paper has shown that the neural network approach to the parameterized expectation solution is a useful alternative to polynomial expansion, and a sensible way to proceed is to estimate the parameters of the parameterized expectations through a combined genetic algorithm/gradient descent method. The results for high values of the variance of the productivity shock, and for a constant relative risk aversion coefficient of .5 appear to be quite reasonable, and indicate that this model should be used at least as a check on other methods. In more complicated models, e.g. multisectoral models or

models with a higher degree of nonlinearity, the genetic algorithm/gradient descent estimation scheme that we propose might be an even more useful approximation tool.

Appendix I: A Mathematical Exposition of the Genetic Algorithm

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

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

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

where $\hat{\theta}$ is the parameter vector of k elemnts,
k the total number of parameters to be estimated.

The genetic algorithm consists of the following seven steps.

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

$$\begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ . \\ . \\ \hat{\theta}_k \end{bmatrix}_1 \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ . \\ . \\ \hat{\theta}_k \end{bmatrix}_2 \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ . \\ . \\ \hat{\theta}_k \end{bmatrix}_i ,...., \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ . \\ . \\ \hat{\theta}_k \end{bmatrix}_p$$

where p is an even number. The population of size k by p may be created with a normal random-number generator. 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 randomly two members of the population: random number, matrices i

$$\begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ . \\ . \\ \hat{\theta}_k \end{bmatrix}_i \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ . \\ . \\ \hat{\theta}_k \end{bmatrix}_j$$

Allow these two vectors, i and j, to breed and create two children, with probability φ (usually φ is close to unity). Coefficient vectors i and j become P(i) and P(j). Again call on a uniform random number ν, multiply it by k, the number of coefficients, and round off at the nearest integer, k*, where k* = round(ν.k). The integer k* becomes the "cut-point" in the breeding process.

Once k* is determined, allow two children to be reproduced. The first k* elements of child 1 of couple (ij), C1(ij), has the first k* elements of parent i, and elements (k*+1 through k) from parent j. Similarly, child C2(ij) has the first k* elements from parent j, and the last (k*+1 through k) elements from parent i. The elements from parents i and j are called "chromosomes". The children C1 and C2 from parents ij appear as follows:

$$\begin{bmatrix} \hat{\theta}_{i,1} \\ \hat{\theta}_{i,2} \\ \\ \hat{\theta}_{i,k^*} \\ \hat{\theta}_{j,k^*+1} \\ . \\ . \\ \hat{\theta}_{j,k} \\ \end{bmatrix}_{C1,ij} \begin{bmatrix} \hat{\theta}_{j,1} \\ \hat{\theta}_{j,2} \\ \\ \hat{\theta}_{j,k^*} \\ \hat{\theta}_{i,k^*+1} \\ . \\ . \\ \hat{\theta}_{i,k} \\ \end{bmatrix}_{C2,ij}$$

Step three: Each element or chromosome of the two children is subject to a mutation. The process takes place as follows:

(1) define the mutation probability μ, usually μ= 1/k
(2) generate a uniform random number, ν, 2k times, corresponding to the elements of both children
(3) if ν < μ, set parameter δ = 1 for element i, other, δ =0, for all i.
(4) for each child, for each element, generate a random normal variable: ψ. We thus have each element i being "mutated: by a factor ψ_i . The children become:

$$\begin{bmatrix} \hat{\theta}_{i,1} + \delta_{1,1}\psi_{1,1} \\ \hat{\theta}_{i,2} + \delta_{1,2}\psi_{1,2} \\ \\ \hat{\theta}_{i,k^*} + \delta_{1,k^*}\psi_{1,k^*} \\ \hat{\theta}_{j,k^*+1} + \delta_{1,k^*+1}\psi_{1,k^*+1} \\ . \\ . \\ \hat{\theta}_{j,k} + \delta_{1,k}\psi_{1,k} \\ \\ \end{bmatrix}_{C1,ij} \begin{bmatrix} \hat{\theta}_{j,1} + \delta_{2,1}\psi_{2,1} \\ \hat{\theta}_{j,2} + \delta_{2,2}\psi_{2,2} \\ \\ \hat{\theta}_{j,k^*} + \delta_{2,k^*}\psi_{2,k^*} \\ \hat{\theta}_{i,k^*+1} + \delta_{2,k^*+1}\psi_{2,k^*+1} \\ . \\ . \\ \hat{\theta}_{i,k} + \delta_{2,k}\psi_{2,k} \\ \\ \end{bmatrix}_{C2,ij}$$

Step four: tournaments. The four members of the "family" engage in a tournament. They are evaluated by the fitness criterion, given by the objective function. The two lowest, whether parents or children, survive and pass to the next generation, while the two highest are extinguished.

Step five: repeat process, with parents i and j returning to the population pool for possible selection again, until the next generated is populated with p coefficient vectors.

Step six: 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 this member from the older generation displaces the worst member of the new generation, and is eligible for selection for cross over in the coming generation.

Step seven: continue the process for many generations. Evaluate convergence by the behavior of the best member of each generation, according to the fitness criterion. If there is little change in the fitness evaluation of the best member of each passing generation for 50 generations, one can assume that the genetic search has converged to this element.

References

Arifovic, J. (1994), "Genetic Algorithm Learning and the Cobweb Model," *Journal of Economic Dynamics and Control* 18, 3-28.

Back, T. (1993), "Optimal Mutation Rates in Genetic Search," in S. Forrest, ed., Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, Calif: Morgan-Kaufman, 2-8.

Beaumont, P.M. and P.T. Bradshaw (1995), "A Distributed Parallel Genetic Algorithm for Solving Optimal Growth Models,'' *Computational Economics* 8, 159-179.

Bishop, C. M. (1995), Neural Networks for Pattern Recognition. New York: Oxford University Press.

Chyi, Y.L. (1997), "Solving the Stochastic Growth Model by a Neural Network Learning Approach". Working Paper, Department of Economics, National Tsing Hua University, Taiwan.

Cooley, T.F. and G.D.Hansen (1989), "The Inflation Tax in a Real Business Cycle Model," *American Economic Review* 79, 733-748.

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. Cambridge, Mass.: MIT Press.

Holland, J. (1975), Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.

Marcet, A. (1993), "Simulation Analysis of Dynamic Stochastic Models: Applications to Theory and Estimation," Working Paper, Universitat Pompeu Fabra.

Mitchell, M. (1996), An Introduction to Genetic Algorithms. Cambridge: MIT Press.

Sargent, T. J. (1993), Bounded Rationality in Macroeconomics. New York: Oxford University Press.

Semmler, W. and G. Gong (1997), "A Numerical Procedure to Estimate Real Business Cycle Models Using Simulated Annealing," in H. Amman et al., ed., Computational Approaches to Economic Problems, London: Kluwer.

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.

_____ 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. Ulig (1990), "Solving Nonlinear Stochastic Growth Models: A Comparison of Alternative Solution Methods," *Journal of Business and Economic Statistics* 8, 1-17.

Uhlig, H. (1997), "A Toolkit for Analyzing Nonlinear Dynamic Stochastic Models Easily," Working Paper, CentER, University of Tilburg.

Table I						
Benchmark Values for the Exact Solution						
Exact Solution						
beta	sigma:	~~Volatility~~	~~D-M Stat~~	~~TR-Stat~~	~~R-SQ~~	~~I/C Ratio~~
0.95	**0.02**	0.01	146.04	9.07	0.01	2.34
0.95	**0.10**	0.08	125.33	9.36	0.01	2.03
0.98	**0.02**	0.01	173.72	9.07	0.01	2.56
0.98	**0.10**	0.08	170.63	9.23	0.01	2.22

TABLE II: Consumption Volatility Measures

POLYNOMIAL MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.5**	**1.5**	**3**	**0.5**	**1.5**	**3**
0.95	**0.02**	0.01	0.10	0.01	0.01	0.15	0.16
0.95	**0.10**	137610.91	0.09	na	0.04	0.15	na
0.98	**0.02**	0.01	0.03	0.59	0.01	0.10	0.63
0.98	**0.10**	24197.95	0.09	na	0.05	0.10	na

NEURAL NETWORK MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.50**	**1.50**	**3.00**	**0.50**	**1.50**	**3.00**
0.95	**0.02**	0.01	0.01	0.69	0.01	0.03	0.70
0.95	**0.10**	0.01	0.07	na	0.02	0.08	na
0.98	**0.02**	0.00	0.01	0.01	0.01	0.04	0.72
0.98	**0.10**	0.01	0.09	na	0.13	0.11	na

TABLE III: Den-Haan and Marcet Statistics

POLYNOMIAL MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.5**	**1.5**	**3**	**0.5**	**1.5**	**3**
0.95	**0.02**	396.99	396.37	396.87	385.93	396.50	396.57
0.95	**0.10**	218.57	393.12	na	395.29	395.93	na
0.98	**0.02**	396.29	396.88	393.32	300.24	396.59	391.13
0.98	**0.10**	296.92	390.38	na	391.47	391.09	na

NEURAL NETWORK MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.50**	**1.50**	**3.00**	**0.50**	**1.50**	**3.00**
0.95	**0.02**	396.79	396.98	396.20	388.99	397.00	396.14
0.95	**0.10**	392.77	396.24	na	384.81	396.30	na
0.98	**0.02**	396.92	396.98	396.98	365.04	396.99	396.11
0.98	**0.10**	391.72	396.30	na	340.69	396.73	na

TABLE IV: TR-Squared Statistics

POLYNOMIAL MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.5**	**1.5**	**3**	**0.5**	**1.5**	**3**
0.95	**0.02**	22.11	16.22	na	13.05	14.92	7.84
0.95	**0.10**	16.24	12.75	na	10.96	9.85	na
0.98	**0.02**	12.39	10.61	17.44	13.95	11.08	16.97
0.98	**0.10**	na	17.41	na	14.53	16.37	na

NEURAL NETWORK MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.50**	**1.50**	**3.00**	**0.50**	**1.50**	**3.00**
0.95	**0.02**	11.40	15.03	19.26	13.21	10.89	19.26
0.95	**0.10**	13.28	17.68	na	13.96	17.32	na
0.98	**0.02**	11.46	17.12	15.21	11.11	14.06	9.24
0.98	**0.10**	13.29	13.11	na	11.76	13.47	na

TABLE V: R-Squared Statistics

POLYNOMIAL MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.5**	**1.5**	**3**	**0.5**	**1.5**	**3**
0.95	**0.02**	0.99	0.81	0.04	0.08	0.89	0.31
0.95	**0.10**	0.39	0.26	na	0.20	0.23	na
0.98	**0.02**	0.17	0.06	1.00	0.12	0.26	1.00
0.98	**0.10**	0.50	0.65	na	0.13	0.64	na

NEURAL NETWORK MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.50**	**1.50**	**3.00**	**0.50**	**1.50**	**3.00**
0.95	**0.02**	0.04	0.99	1.00	0.06	0.95	0.99
0.95	**0.10**	0.22	0.89	na	0.28	0.78	na
0.98	**0.02**	0.74	0.98	0.68	0.27	0.05	0.10
0.98	**0.10**	0.22	0.93	na	0.83	0.93	na

TABLE VI: Investment/Consumption Volatility Ratio

POLYNOMIAL MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.5**	**1.5**	**3**	**0.5**	**1.5**	**3**
0.95	**0.02**	0.14	34.73	0.00	2.83	0.89	1.66
0.95	**0.10**	80.89	1.93	na	130.92	0.58	na
0.98	**0.02**	56.19	0.33	0.12	2.03	2.05	0.11
0.98	**0.10**	0.00	8.09	na	71.61	8.07	na

NEURAL NETWORK MODEL

		GENETIC ALGORITHM			GA/GRADIENT DESCENT		
		CRRA:			CRRA:		
beta	sigma:	**0.50**	**1.50**	**3.00**	**0.50**	**1.50**	**3.00**
0.95	**0.02**	444.17	10.99	0.11	3.10	1.24	0.11
0.95	**0.10**	1292.72	10.65	na	583.55	8.77	na
0.98	**0.02**	1425.95	12.26	336.74	1313.14	0.61	4.99
0.98	**0.10**	892.01	6.41	na	0.53	3.37	na

	Table VII: Comparison with Alternative Methods under Low Variance Assumption: sigma = .02												
	Panel A: beta = .95				Panel B:beta = .95				Panel C:beta = .95			
		tau = .5				tau = 1.5				tau = 3		
	D-M Stat	TR-Stat	R-SQ	I/C Ratio	D-M Stat	TR-Stat	R-SQ	I/C Ratio	D-M Stat	TR-Stat	R-SQ	I/C Ratio
Duffy/McNelis-NN	382.71	9.26	0.52	0.08	327.94	11.59	0.03	na	77.77	8.25	0.65	0.50
Duffy/McNelis-PA	27.91	12.43	0.51	398.00	24.65	12.71	0.32	na	17.00	10.90	0.49	na
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
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
	Panel D:beta = .98				Panel E:beta = .98				Panel F:beta = .98			
		tau =0.5				tau=1.5				tau = 3		
	D-M Stat	TR-Stat	R-SQ	I/C Ratio	D-M Stat	TR-Stat	R-SQ	I/C Ratio	D-M Stat	TR-Stat	R-SQ	I/C Ratio
Duffy/McNelis-NN	274.70	9.87	0.57	0.06	10.55	26.11	0.10	na	10.73	13.49	0.15	na
Duffy/McNelis-PA	197.43	16.29	0.51	0.17	369.16	12.73	0.37	na	396.99	8.26	0.58	na
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
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

	Table VIII: Comparison with Alternative Methods								
	High Variance Assumption: sigma = .1								
	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	245.71	14.55	0.10	1.97		365.60	9.99	0.31	na
Duffy/McNelis-PA	76.13	na	0.50	0.00		136.15	11.96	0.16	na
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
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	101.26	13.86	0.12	2.21		22.37	12.83	0.08	na
Duffy/McNelis-PA	156.16	10.31	0.43	2.93		373.86	na	0.03	na
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
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