Solving Non-Linear Rational Expectations Models

Paul L. Fackler*

November 18, 2005

Abstract

A general framework for describing non-linear rational expectation models is developed that involves state variables, response variables and expectation variables. The solution to such models can be expressed in terms of a response function or an expectation function. Computational methods for solving such models involve approximating one of these functions. Alternative methods are characterized by how the expectation operator is approximated, what family of approximation is used for the solution function, what criteria are used for choosing approximation parameters and what algorithm is used to identify the parameters. A user-friendly MATLAB procedure that incorporates a wide variety of possible choices is described.

Keywords: rational expectations, projection methods

*The author is Associate Professor in the Department of Agricultural and Resource Economics at North Carolina State University.

Mail: Paul L. Fackler
Department of Agricultural and Resource Economics
NCSU, Box 8109
Raleigh NC, 27695, USA

e-mail: paul_fackler@ncsu.edu

Web-site: http://www4.ncsu.edu/∼pfackler/

© 2005, Paul L. Fackler

Solving Non-Linear Rational Expectations Models

This paper documents the set of MATLAB based tools for solving nonlinear rational expectations models discussed in Fackler. A general framework for rational expectations models is discussed in the next section and illustrated with two examples. In the following section solution approaches are described. Then a user friendly computational solution procedure written in the MATLAB programming language is described, followed by a section which illustrates the use of the code using one of the examples. An appendix discusses the related topics of linearizing nonlinear models and solving the resulting linear models.

Nonlinear rational expectations present difficulties because their solutions are functions of unknown form and because the equilibrium conditions involve integrals that generally admit no explicit solutions. Numerical solutions typically involve replacing the integral operator by a finite dimensional approximation and replacing the unknown solution function with an approximating function that can be defined in terms of a finite number of parameters.

Following Judd, there are four basic choices that must be made when solving nonlinear rational expectations models. First, how the expectation operator is approximated. Second, what family of approximating functions is used to represent the solution. Third, what criteria are used to determine how parameters of the approximating function are selected. Fourth, what algorithm is used to obtain these parameters.

The MATLAB solvers discussed here have a number of options for each of these choices. The code itself is distributed as part of the COMPECON Toolbox, which was developed to accompany the text by Miranda and Fackler and which is available at `http://www4.nscu.edu/~pfackler/compecon`.

1 Rational Expectations Models

A general framework for describing rational expectations models begins by dividing model variables into state variables, denoted by the d-vector s, and response variables, denoted by the m-vector x (a summary of terms is provided in Table 1). The state variables are distinguished by the existence of an explicit and known updating rule

$$\dot{s} = g(s, x, e),$$

where e is a q-vector of i.i.d shocks and "˙" indicates the next period's value. The state process in this canonical form is first order in time. Models with higher order lags, however, can always be redefined as first order by expanding the dimension of the state space.

The response variables satisfy an equilibrium relationship of the form

$$f(s, x, z) = 0,$$

where

$$z = E\Big[h(s, x, e, \dot{s}, \dot{x})\Big]$$

the expectation being taken with respect to e. Here $f : R^{d+m+p} \to R^m$ and $h : R^{d+m+q+d+m} \to R^p$. Collectively, g, f and h define a model.

A solution to the model is a response function $x = \Theta(s)$ such that

$$f\left(s, \Theta(s), E\Big[h(s, \Theta(s), e, \dot{s}, \Theta(\dot{s}))\Big]\right) = 0$$

for all s, with

$$\dot{s} = g(s, \Theta(s), e).$$

The existence of the function $x = x(s, z)$ that satisfies $f(s, x(s, z), z) = 0$ for all s and z is assumed (there need not exist an explicit form for $x(s, z)$ although it is convenient when one is available). An alternative characterization of the equilibrium is

$$\Theta(s) = x\Big(s, E\Big[h(s, \Theta(s), e, \dot{s}, \Theta(\dot{s}))\Big]\Big).$$

The equilibrium can also be characterized in terms of an expectation function $z = \Psi(s)$ that solves

$$\Psi(s) = E\left[h\left(s, x(s, \Psi(s)), e, \dot{s}, x\Big(\dot{s}, \Psi(\dot{s})\Big)\right)\right]$$

for all s, with

$$\dot{s} = g(s, x(s, \Psi(s)), e).$$

Before proceeding, two simple examples should help make the setup concrete. The first example is a standard stochastic growth model in which the capital stock K evolves according to

$$\dot{K} = e^V K^\beta + \gamma K - c,$$

where c is consumption and V is a technology shock that evolves according to

$$\dot{V} = \rho V + e,$$

with $e \sim N(0, \sigma^2)$. Given a representative agent with utility $c^{1-\alpha}/(1-\alpha)$ and discount factor δ, the equilibrium condition for the growth model is

$$c^{-\alpha} - \delta E\Big[\dot{c}^{-\alpha}(\beta e^{\dot{V}} \dot{K}^{\beta-1} + \gamma)\Big] = 0.$$

The deterministic steady state (with e identically equal to 0) for this model is $\bar{V} = 0$,

$$\bar{K} = \left(\frac{1/\delta - \gamma}{\beta}\right)^{1/(\beta-1)}$$

and $\bar{c} = \bar{K}^{\beta} + (\gamma - 1)\bar{K}$.

In the general notation, s is two dimensional, composed of K and V, and there is a single response variable c, a single shock e and a single expectation variable. The functions g, h, f and x are

$$g(s, x, e) = [e^{s_2} s_1^{\beta} + \gamma s_1 - x \quad \rho s_2 + e], \tag{1}$$

$$h(s, x, e, \dot{s}, \dot{x}) = \dot{x}^{-\alpha}(\beta e^{\dot{s}_2} \dot{s}_1^{\beta-1} + \gamma) \tag{2}$$

$$f(s, x, z) = x^{-\alpha} - \delta z \tag{3}$$

and

$$x(s, z) = (\delta z)^{-1/\alpha}. \tag{4}$$

The second example is Lucas' well known asset pricing model. An economy has available a set of stocks, the ith of which pays dividend d_i. The dividends each follow first order autoregressive processes

$$\dot{d}_i = \mu_i + \theta_i(d_i - \mu_i) + e_i$$

where $e \sim N(0, \Sigma)$. A representative agent consumes the dividends, generating utility of $c^{(1-\gamma)}/(1-\gamma)$ in each period, where $c = \sum d_i$. Future expected utility is discounted using discount factor δ. The price of the ith asset p_i satisfies

$$\left(\sum d_i\right)^{-\gamma} p_i = \delta E\left[\left(\sum \dot{d}_i\right)^{-\gamma} (\dot{p}_i + \dot{d}_i)\right].$$

The deterministic steady state (with e identically equal to 0) for this model is $\bar{d} = \mu$ and $\bar{p} = [\delta/(1-\delta)]\mu$.

In the general notation, the state vector is the vector of dividends d and the response vector is the vector of asset prices p. The state transition equation is

$$g(s, x, e) = \mu + \theta(s - \mu) + e. \tag{5}$$

and the expectation function is

$$h(s, x, e, \dot{s}, \dot{x}) = (\underline{1}^\top \dot{s})^{-\gamma}(\dot{x} + \dot{s}). \tag{6}$$

where $\underline{1}$ is a vector of ones. The equilibrium relationship is

$$f(s, x, z) = (\underline{1}^\top s)^{-\gamma} x - \delta z = 0, \tag{7}$$

or, alternatively,

$$x(s, z) = (\underline{1}^\top s)^{\gamma} \delta z. \tag{8}$$

Table 1: Notation Summary

Variables	
$s \in S \subseteq R^d$	state variables
$x \in [a, b] \subseteq R^m$	response variables
$z \in R^p$	expectation variables
$e \in R^q$	shocks
Specified Functions	
$g : R^{d+m+q} \to R^d$	state transition
$h : R^{d+m+q+d+m} \to R^p$	expectation
$f : R^{d+m+p} \to R^m$	equilibrium
$x : R^{d+p} \to R^m$	explicit equilibrium response
Solution Functions	
$\Theta : R^d \to R^m$	response
$\Psi : R^d \to R^p$	expectation

2 Numerical Solution Approaches

There are four sets of issues that must be addressed in solving nonlinear rational expectations models. First, how the expectation operator is approximated. Second, what family of approximating functions is used to represent the solution. Third, what criteria are used to determine how parameters of the approximating function are selected. Fourth, what algorithm is used to obtain these parameters.

It is assumed that expectations can be represented well by a discrete distribution in which the shock process takes on value e_j with probability w_j:

$$E[f(e)] \approx \sum_j w_j f(e_j).$$

This framework is general enough to handle most of the common approaches to numerically approximating integrals including Newton-Cotes and Gaussian quadrature as well as Monte Carlo and quasi-Monte Carlo methods (for which the weights are typically the reciprocal of the number of values used).

The functional forms of Θ and Ψ are, in general, unknown and so must be approximated. It is convenient to use families of approximating functions that are linear in a set of coefficients, i.e., functions of the form $\phi(s)\theta$ or $\phi(s)\psi$, where $\phi(s)$ is a vector of n basis functions and where θ and ψ are $n \times m$ and $n \times p$ matrices of coefficients. Polynomials and polynomial splines (including piecewise linear functions) fall into this class. The approximating function will typically be defined on some subset S of the state space, particularly when the state space is unbounded.

If the response function is approximated using $\Theta(s) \approx \phi(s)\theta$, define the residual function

$$r(s,\theta) = f\left(s, \phi(s)\theta, \sum_j w_j h\Big(s, \phi(s)\theta, e_j, \dot{s}_j, \phi(\dot{s}_j)\theta\Big)\right), \tag{9}$$

with

$$\dot{s}_j = g(s, \phi(s)\theta, e_j).$$

Alternatively, the residual function can be defined using $x(s,z)$:

$$r(s,\theta) = \phi(s)\theta - x\left(s, \sum_j w_j h\Big(s, \phi(s)\theta, e_j, \dot{s}_j, \phi(\dot{s}_j)\theta\Big)\right). \tag{10}$$

If the expectation function is approximated using $\Psi(s) \approx \phi(s)\psi$, the residual function is defined as

$$r(s,\psi) = \phi(s)\psi - \sum_j w_j h\left(s, x(s, \phi(s)\psi), e_j, \dot{s}_j, x\Big(\dot{s}_j, \phi(\dot{s}_j)\psi\Big)\right), \tag{11}$$

with

$$\dot{s}_j = g(s, x(s, \phi(s)\psi), e_j).$$

The residual function should be close to zero and a variety of criteria for picking the parameter matrix $c = \theta$ or ψ are available to ensure this, including the Galerkin, collocation and least-squares methods. The Galerkin method solves

$$\int_S \phi(s) r(s, c) ds = 0.$$

As there are n basis functions comprising $\phi(s)$, this gives n equations in n unknowns. The collocation method solves

$$r(s_i; c) = 0$$

at n values of s. The least squares method solves

$$\min_c \int_S r^2(s, c) ds.$$

The Galerkin and the least squares methods necessitate evaluating an integral which typically does not have an explicit solution. In practice the integral would therefore be replaced by

$$\sum_i \omega_i \phi(s_i) r(s_i, c) = 0$$

or

$$\min_c \sum_i \omega_i r^2(s_i, c)$$

for some selected values s_i and weights ω_i. To keep the discussion manageable, only the collocation approach, which avoids the need to evaluate an additional integral, is discussed.

The collocation approach requires that a set of n nodal values be selected. One can then form the $n \times n$ matrix Φ, each row of which is composed of one of the $\phi(s)$. This results in a system of nonlinear equations of order nm if the response function is approximated and np if the expectation function is approximated.

Several methods exist to solve the system of equations for the parameter values. One approach is to use general purpose rootfinding algorithms. Newton's method uses the iteration

$$c^{(k+1)} = c^{(k)} - \alpha \Big[r_c(s, c^{(k)}) \Big]^{-1} r(s, c^{(k)}).$$

Here $r_c(s, c^{(k)})$ is the $nm \times nm$ or $np \times np$ matrix of partial derivatives of the residual function, evaluated at the n state nodes, with respect to the parameter values. Newton's method requires a linear solve at each iteration of dimension nm or np. In addition, it requires that the user code derivatives or that numerical derivatives be

employed. In particular, the partial derivatives g_x, h_x, $h_{\dot{s}}$ and $h_{\dot{x}}$ must be coded, as well as (f_x,f_z), (x_z) or (x_s,x_z) for the first or second response approximation approach or the expectation approximation approach, respectively.

An alternative is Broyden's method, which uses the iteration

$$c^{(k+1)} = c^{(k)} - \alpha B^{(k)} r(s, c^{(k)}).$$

where $B^{(k)}$ is an approximation to $[r_c(s, c^{(k)})]^{-1}$. An $nm \times nm$ or $np \times np$ matrix must still be computed at each iteration, but no linear solve is performed. Furthermore, $B^{(k+1)}$ is a rank-one update of $B^{(k)}$ and hence can be computed easily, so long as the problem is not too large.

In addition to generic rootfinding algorithms, the structure of the problem suggests two natural fixed point iteration schemes using (10) or (11). If the response function is approximated, let $\theta^{(k)}$ denote the value of θ at the start of the kth iteration. The iteration proceeds as follows:

$$\begin{aligned} x &= \phi(s)\theta^{(k)} \\ \dot{s}_j &= g(s, x, e_j) \\ \dot{x}_j &= \phi(\dot{s}_j)\theta^{(k)} \\ z &= \textstyle\sum_j w_j h(s, x, e_j, \dot{s}_j, \dot{x}_j) \\ \tilde{\theta} &= \Phi^{-1} x(s, z) \\ \theta^{(k+1)} &= \theta^{(k)} + \alpha(\tilde{\theta} - \theta^{(k)}) \end{aligned}$$

for some value of α. This iteration is continued until the change in one or more of θ, x or z is sufficiently small from one iteration to the next.

If the expectation function is approximated, let $\psi^{(k)}$ denote the value of ψ at the start of the kth iteration. The iteration proceeds as follows:

$$\begin{aligned} x &= x(s, \phi(s)\psi^{(k)}) \\ \dot{s}_j &= g(s, x, e_j) \\ \dot{x}_j &= x\Big(\dot{s}_j, \phi(\dot{s}_j)\psi^{(k)}\Big) \\ z &= \textstyle\sum_j w_j h(s, x, e_j, \dot{s}_j, \dot{x}_j) \\ \tilde{\psi} &= \Phi^{-1} z \\ \psi^{(k+1)} &= \psi^{(k)} + \alpha(\tilde{\psi} - \psi^{(k)}) \end{aligned}$$

for some value of α. Again, the iteration is continued until some convergence criterion is met.

The parameter α is a step size parameter that can be set to a value less than 1 if the algorithm is unstable. A value larger than 1 may speed convergence for a stable problem. The optimal value of α is problem specific and may require experimentation to determine (see Judd, pp. 78-80).

Fixed point iterations of this sort are not guaranteed to converge even with good starting values and convergence is at best linear, in contrast to the quadratic and superlinear rates of convergence for Newton's and Broyden's methods, respectively. The fixed point iterations described above have some advantages over general rootfinding approaches in that the amount of work and the memory requirements required to perform an iteration are kept to a minimum. Furthermore, for large nm or np the linear solve in Newton's method may become unacceptably slow. Memory limitations may prevent an $nm \times nm$ or $np \times np$ matrix from even being formed unless the derivatives can be represented in sparse form. Furthermore, sparsity does not help when using Broyden's method, as the rank one matrix updating rule does not preserve sparsity.

The only linear solve used in the fixed-point iterations is the n-dimensional linear solve involved in updating $\tilde{\theta}$ or $\tilde{\psi}$. Furthermore, if tensor product bases are used, this can be solved in terms of the tensor product of the inverses of basis matrices for individual states, making the iterative approach possible even for relatively high dimensional problems.

Ultimately, the choice of algorithm depends on a number of factors. Although the amount of work per iteration is important, the number of iterations needed to successfully solve the problem is also important. Fixed point iterations typically require a large number of iterations and are generally less reliable. Furthermore, in practice, the main computational work at each iteration is the evaluation of the $\phi(\dot{s}_j)c$, which is required by any method. Thus the amount of work per iteration used by Broyden's method is not much greater than with the fixed-point iteration schemes.

If a reasonable starting value for A can be identified, Broyden's method will tend to use fewer iterations and hence provide superior performance. Using the inverse of the Jacobian of the residual function is a good choice but this can be time consuming to compute, especially for problems with multidimensional states and/or with a large number of shock values. To reduce the computational burden and still provide a good starting value, the Jacobian of the residual function for the deterministic model (with a single "shock" set equal to $E[e]$) can be used.

The Jacobian of the residual function can be approximated numerically or analytically if the partial derivatives of f (or x), g and h are available. Specifically, if the response function is approximated the Jacobian of the residual function is

$$\left(f_x + f_z[h_x + (h_{\dot{s}} + h_{\dot{x}}\phi'(\dot{s})\theta)g_x]\right) \otimes \phi(s) + f_z h_{\dot{x}} \otimes \phi(\dot{s})$$

or, if an explicit form for $x(s,z)$ exists,

$$\left(I_m - x_z[h_x + (h_{\dot{s}} + h_{\dot{x}}\phi'(\dot{s})\theta)g_x]\right) \otimes \phi(s) - x_z h_{\dot{x}} \otimes \phi(\dot{s})$$

An analogous expression for expectation function approximation is

$$\left(I_p - [h_z + (h_{\dot{s}} + h_{\dot{x}}[x_s(\dot{s},\dot{z}) + x_z(\dot{s},\dot{z})\phi'(\dot{s})\psi])g_x]x_z\right) \otimes \phi(s) - h_{\dot{x}}x_z(\dot{s},\dot{z}) \otimes \phi(\dot{s})$$

It is straightforward to extend the framework to cases with inequality constraints on the response variables. Suppose that $a \leq x \leq b$ and

$$x_i > a_i \Rightarrow f_i(s, x, z) \geq 0$$

$$x_i < b_i \Rightarrow f_i(s, x, z) \leq 0.$$

No changes need be made in the solution algorithm if an explicit solution of the form $x = x(s, z)$ exists that satisfies these conditions. If no explicit solution exists, however, the equilibrium condition can be converted to a standard rootfinding problem using

$$\min(\max(f(s, x, z), a - x), b - x) = 0$$

or an analogous semi-smooth transformation (Miranda and Fackler, chapter 3).

In models for which an explicit function $x(s, z)$ does not exist, the residual function (9) can always be used in conjunction with a standard rootfinding algorithm. If the other approaches are attempted, one could use a standard rootfinding algorithm to determine the values of x that solve

$$f(s, x, z) = 0$$

at specific values of s and z. With n nodal values of s, this involves n separate solves, each of m equations in m unknowns.

In addition, if the expectation function is approximated and if h is a function of $\dot{x}$, as is typically the case, the value of $\dot{x}_j$ that solves $f(\dot{s}_j, \dot{x}_j, \phi(\dot{s}_j)\psi)$ must also be found. Rather than go to the expense of performing these calculations, it is easier and faster to interpolate using the values (s, x) that have already been found. Linear interpolation could be used or, if the response function can be reasonably approximated with a function of the form $\phi(s)c_x$, one could compute

$$c_x = \Phi^{-1}x$$

and use

$$\dot{x}_j = \phi(\dot{s}_j)c_x.$$

This extra trouble may be worthwhile if the expectation function has relatively low dimension and/or it can be approximated more accurately with a lower order approximation than the response function can.

3 MATLAB Implementation

Given the modeling framework described above, it is possible to develop a general purpose procedure for solving rational expectations model that requires a user simply to specify the model and the choices of numerical methods without having to worry too much about algorithmic details. This section describes a MATLAB implementation, `resolve`, of such a procedure. The procedure makes extensive use of the COMPECON Toolbox described in Miranda and Fackler, which can be downloaded from the author's website.

The calling syntax for `resolve` is

```
[c,scoord,x,z,f,resid] = resolve(model,fspace,options,cinit);
```

The first three inputs are structure variables[1] that define, respectively, the nature of the model, the nature of the family of approximating functions and the specific solution options desired. The fourth input is a matrix of initial coefficients (either $n \times m$ or $n \times p$).

The `model` variable has the following named fields

`func`	the name of the model function file (described below)
`params`	additional parameters to be passed to the model function file
`e`	a discretization of the shock variables ($K \times q$)
`w`	probabilities associated with the shocks ($K \times 1$)
`explicit`	0/1 variable, 1 if $x(s, z)$ has an explicit form
`noxnext`	0/1 variable, 1 if h is not a function of $\dot{x}$

The model is described by the functions g, h, f and $x(s, z)$, which should be coded in the file specified by the `func` field (details are given below). Any model parameters used to evaluate these functions should be collected in the cell array in the `params` field.[2] The possible shocks should be specified in a $K \times q$ matrix and stored in the `e` field. The K-vector of probabilities associated with the shocks should be stored in the `w` field (the COMPECON Toolbox has several routines for computing discrete approximations for several commonly used multivariate distributions, including the normal and lognormal). If an explicit solution to $x(s, z)$ is coded and returned by the `func` file, set the `explicit` field to 1. The most time-consuming part of the algorithm is the computation of $\dot{x}$, used in evaluating h. If h is not a function of $\dot{x}$, the `noxnext` field should be set to 1.

[1] Structure variables are user defined data structures that have fields that can be accessed by name.

[2] A cell array is essentially an array of pointers to other variables. This data type allows any sort of data to be bundled together as a single variable.

The `fspace` variable is defined by the CompEcon Toolbox function `fundefn` (or `fundef`); for details refer to Miranda and Fackler (chap.6). Suffice it to say here that the toolbox allows users to easily specify polynomials (using the Chebyshev basis) and splines (using B-spline bases). Once a family of functions is defined, it can be evaluated using `funeval`. Supporting functions for computing standard nodes and basis matrices and for fitting interpolating or smoothing functions are also available.

The `options` variable may have any of the following fields

`expapprox`	1 if the expectation function is approximated, 0 otherwise
`usebroyden`	1 if Broyden's method is desired, 0 for fixed point iteration
`lowmemory`	1 to use less memory (typically executes more slowly)
`stepsize`	the α parameter in the fixed-point coefficient update
`tol`	convergence criteria
`maxit`	maximum number of iterations allowed
`showiters`	1 to dispay summary information at each iteration
`checks`	1 to check whether next period's state is ever outside the approximation bounds
`nres`	inflation factor used when computing residuals
`xtol`	convergence criteria for computing x
`xmaxit`	maximum iterations to compute x
`lcpmethod`	method used to compute x ('minmax' or 'smooth')

The solution procedure supports both response and expectation approximation. Set the `expapprox` field to 0 for the former and 1 for the latter (the default is 1). Both fixed-point iteration and Broyden's method are supported. Set the `usebroyden` field to 0 for the former and 1 for the latter (the default is 0). Broyden's method requires that the inverse Jacobian approximation (B) be initialized in some way. The code here uses a numerical approximation to the Jacobian for the deterministic model. This provide a relatively quick but fairly high quality approximation. An alternative that requires extra coding on the part of the user is to code the partial derivatives of the model's functions (this alternative is described more fully below).

If memory in limited, set `lowmemory` to 1; this will result in more calls to the model function file and therefore slow execution. Set the `stepsize` field to a number less than 1 if a fixed-point procedure is used and appears to be unstable and to a number greater than 1 to attempt to accelerate convergence (the default value is 1). Iterations will stop when the maximal absolute change in the coefficients is less than the value specified in the `tol` field (the default is the square root of machine precision, approximately 10^{-8} on most machines). They will also stop if the number of iterations reaches the value specified in the `maxit` field (the default is 500). Summary

information for each iteration is displayed if the `showiters` field is non-zero (the default is 0).

The appropriate approximation bounds for the state space is often difficult to determine. Setting the option `checks` equal to 1 causes `resolve` to check whether next period's state requires extrapolation beyond the approximation bounds. If so, a warning message is printed. This does not necessarily indicate a poor approximation, especially for values of the state in the interior of the approximation region. It does, however, suggest that the results should be examined carefully.

If the user requests that residuals are computed (by requesting 6 outputs), the field `nres` determines how many points are evaluated. The number of nodal points used for each state variable in computing the solution will be multiplied by this value (the default is 10). The residuals provide a measure of the quality of the approximation and it is useful to evaluate them at many non-nodal points.

If $x(s, z)$ does not have an explicit form, `resolve` uses Newton's method to solve $f(s, x, z) = 0$ for $x(s, z)$ (the exception occurs when Broyden's method is used to find a response function approximation, in which case, when $f(s, x, z) = 0$ is used as the residual function rather than $\phi(s)\theta - x(s, z)$). The remaining fields control the behavior of the Newton algorithm. The fields `xtol` and `xmaxit` are analogous to `tol` and `maxit` and have the same default values.

If there are bounds on the response variables, the user may choose to use either the minmax reformulation, i.e.,

$$\min(\max(f(s, x, z), a - x), b - x) = 0$$

or a semi-smooth reformulation (see Miranda and Fackler, chap. 3, for a discussion of semi-smooth reformulations of complementarity problems). The default is `minmax`, which generally is evaluated somewhat more quickly but also has a greater tendency to get stuck.

The first output from `resolve` is the approximation function coefficient matrix c (the $n \times m$ matrix θ if the response approximation is used or the $n \times p$ matrix ψ if the expectation approximation is used). The second output `scoord` is a set of d vectors of values of the state variables, stored as a cell array. The remaining outputs represent the values of x $(N \times m)$, z $(N \times p)$, f $(N \times m)$ and the residuals, either $x - \phi(s)\theta$ or $z - \phi(s)\psi$ (where N is the number of points in the grid formed by the vectors in `scoord`). If the residuals are not computed, the values of `scoord` are the nodal values used to compute the approximation. If the residuals are computed, however, the values of `scoord` are evenly spaced points over the interval of approximation in each dimension.

The user must also code the file named in the `func` field of `model`. A template for this file is:

```
function [out1,out2]=func(flag,s,x,z,e,shat,xhat, ...
                          additional parameters);
  switch flag
  case 'f'
    out1 = f(z,x,z)
    out2 = df(s,x,z)/dx
  case 'x'
    out1 = x(s,z)
  case 'g'
    out1 = g(s,x,e)
  case 'h'
    out1 = h(s,x,e,shat,xhat)
  case 'bounds'
    out1 = a
    out2 = b
  end
```

The first variable, `flag`, passed to the model function file determines the nature of the output that is needed. In each case, an $N \times d$ matrix `s` is passed as the second argument and the last arguments are always the parameters specified in the `params` field of the `model` variable. Other inputs will depend on which value of `flag` is passed. The number N of rows in `s` depends on whether the `lowmemory` option is used or not. If it is used, $N = n$, the number of state nodes and `resolve` loops over the K values of the shocks. If it is not used, $N = nK$, the number of state nodes times the number of shock values.

If `flag` is '`f`', the procedure will be passed `s`, the $N \times m$ matrix `x` and the $N \times p$ matrix `z` and the function should return the $N \times m$ matrix of values of $f(s, x, z)$. If there is no explicit solution to $x(s, z)$, the function should also return $\partial f(s, x, z)/\partial x$, which is used to compute x numerically. The '`f`' flag is only used if there is no explicit solution $x(s, z)$.

When `flag` is '`x`', the model function file will be passed `s` and `z` and should return the $N \times m$ matrix of values of $x(s, z)$. The '`x`' flag is only used if there is an explicit solution $x(s, z)$. It is thus only necessary to code either for the '`f`' or the '`x`' flags.

When the `flag` variable is set to '`g`', the function is passed `s`, `x` and `e`, with latter an $N \times q$ matrix. The function should return the $N \times d$ matrix of values of $g(s, x, e)$. When the `flag` variable is set to '`h`', the function is passed `s`, `x`, `e`, `shat` $(N \times d)$ and `xhat` $(N \times m)$. The function should return the $N \times p$ matrix of values of $h(s, x, e, \dot{s}, \dot{x})$.

The `flag` variable is passed as '`b`' only if there is no explicit solution to $x(s, z)$. The function should return two outputs, both $N \times m$, representing the bounds a and b, where $a(s) \leq x(s) \leq b(s)$.

If Broyden's method is used, significant execution time reductions can be obtained by coding the partial derivatives of x, g and h. Specifically, if `flag` is `x`, the second and third outputs from the model function file should be x_s ($N \times m \times d$) and x_z ($N \times m \times p$). If `flag` is `g`, the second output from the model function file should be g_x ($N \times d \times m$). If `flag` is `h`, the second through the fourth outputs from the model function file should be h_x ($N \times p \times m$), $h_{\dot{s}}$ ($N \times p \times d$) and $h_{\dot{x}}$ ($N \times p \times m$). Analytical derivatives are used automatically if the model function file has four outputs; otherwise numerical derivatives are used.

A few comments are in order. In principle, the software can handle problems with an arbitrary number of variables. In practice, the number of state variables is a limiting factor because the number of coefficients and nodal points needed to approximate arbitrary functions grows exponentially with the number of state variables. This means that memory or patience may run out before a solution can be obtained.

Good starting values will help considerably and may be crucial for convergence for any of the algorithms. In the example presented in the next section, the deterministic steady state is computed and the model is linearized around the steady state. The solution to the linearized model is then used to obtain initial values.

4 Implementing the Growth Model

The growth model has parameters α, β, γ, ρ and δ. The model function file, called `mfre01`, is a straightforward coding of equations (1)-(4):

```
function out=mfre01(flag,s,x,z,e,snext,xnext, ...
                      alpha,beta,gamma,rho,delta);
switch flag
case 'f';
  out = x.^(-alpha)-delta*z;
case 'x'
  out = (delta*z).^(-1./alpha);
case 'g';
  out = [exp(s(:,2)).*s(:,1).^beta+gamma*s(:,1)-x ...
         rho*s(:,2)+e];
case 'h'
  MPC=beta*exp(snext(:,2)).*snext(:,1).^(beta-1)+gamma;
  out=xnext.^(-alpha).*MPC;
end
```

To solve the model, one first defines the parameter values:

```
alpha =  3;        % utility parameter
beta  =  0.33;     % production elasticity
gamma =  1;        % capital survival rate
rho   =  0.9;      % production shock autocorrelation
delta =  0.95;     % discount factor
sigma =  0.02;     % production shock volatility
```

These parameters are among the set of parameters used in the comparisons reported in Taylor and Uhlig.

Next one discretizes the shock process

```
K     = 3;
[e,w] = qnwnorm(K,0,sigma^2);
```

Here the CompEcon Toolbox function `qnwnorm` is used to compute 3-point Gaussian quadrature nodes and weights.

Next the structure variable `model` is created.

```
model.func     = 'mfre01';
model.params   = {alpha beta gamma rho delta};
model.e        = e;
model.w        = w;
model.explicit = 1;
```

To provide starting values, the deterministic steady state is computed and the model is linearized around the steady state. This uses the CompEcon Toolbox function `relin` to compute the linearized model.[3]

```
sstar = [((1/delta-gamma)/beta)^(1/(beta-1)) 0];
xstar = sstar(1)^beta+(gamma-1)*sstar(1);
[C,P] = relin(model,sstar,xstar);
```

The state space is naturally defined on $[0, \infty) \times (-\infty, \infty)$, a space that is impractical for numerical work. The interval $K \in [0.5K^*, 1.5K^*]$ is used here, where K^* is the steady state value obtained above. An appropriate range for the technology shock is given by solving for the minimum and maximum values of $V = \rho V + e$ (i.e., $V = e/(1-\rho)$). The CompEcon Toolbox function `fundefn` is used below to define the `fspace` structure variable that will be passed to `resolve`. The approximation uses a tensor product polynomial basis with order 9 polynomials for K and order 5 polynomials for V (n defines the dimension of the basis for each variable, which in this case is one plus the polynomial order).

[3]Computation of the steady state and model linearization are discussed in an appendix.

```
n = [10 6];
smin = [0.5*sstar(1)  min(e)/(1-rho)];
smax = [1.5*sstar(1)  max(e)/(1-rho)];
fspace = fundefn('cheb',n,smin,smax);
```

The solution options are then defined. The response function is approximated and Broyden's method is used to find the solution.

```
options=struct('usebroyden',1,...
               'expapprox', 0);
```

The last step before calling the solver is to obtain initial values. Here a grid of points in $[K, V]$ is created and the solution to the linearized problem is used to define initial values. Initial coefficient values are then fit to this grid of points. The functions `funnodes`, `gridmake`, `funfitxy` are part of the CompEcon Toolbox.

```
snodes = funnode(fspace);
S      = gridmake(snodes);
xinit  = xstar+(S-sstar(ones(size(S,1),1),:))*C';
c = funfitxy(fspace,snodes,xinit);
```

Finally, the solver is called:

```
[c,s,x,z,f] = resolve(model,fspace,options,c);
```

The file `demre01`, provided with `resolve`, contains the code just described as well as code to generate plots of solution and residual functions. Also provided is a file `demre02`, along with an associated model function file, that implements the asset pricing model described in equations (7)-(8). Thus only two short files are required to obtain a solution to a non-linear rational expectations model.

Appendix: Linearizing the Model

Define $\bar{e} = E[e]$ and let $\bar{s}$ and $\bar{x}$ be the "steady-state" values that satisfy

$$\bar{s} - f(\bar{s}, \bar{x}, \bar{e}) = 0$$

and

$$f(\bar{s}, \bar{x}, h(\bar{s}, \bar{x}, \bar{e}, \bar{s}, \bar{x})) = 0.$$

A linearization around these steady state values is defined by the linear equations

$$\begin{bmatrix} I_n & 0_{mn} \\ -f_z h_{\dot{s}} & -f_z h_{\dot{x}} \end{bmatrix} \begin{bmatrix} \Delta \dot{s} \\ \Delta \dot{x} \end{bmatrix} = \begin{bmatrix} g_s & g_x \\ f_s + f_z h_s & f_x + f_z h_x \end{bmatrix} \begin{bmatrix} \Delta s \\ \Delta x \end{bmatrix},$$

where the partial derivative matrices are evaluated at $\bar{s}$, $\bar{x}$ and $\bar{e}$ and where Δ represents deviations from the steady state, e.g., $\Delta s = s - \bar{s}$.[4]

A solution to this model is an $n \times n$ matrix P such that $\Delta \dot{s} = P \Delta s$ and an $m \times n$ matrix C such that $\Delta x = C \Delta s$. Hence P must satisfy $P = g_s + g_x C$, and

$$-f_z h_{\dot{s}} P - f_z h_{\dot{x}} C P = [f_s + f_z h_s] + [f_x + f_z h_x] C.$$

This can be written in matrix form as

$$\begin{bmatrix} I_n & 0_{mn} \\ -f_z h_{\dot{s}} & -f_z h_{\dot{x}} \end{bmatrix} \begin{bmatrix} P \\ CP \end{bmatrix} = \begin{bmatrix} g_s & g_x \\ f_s + f_z h_s & f_x + f_z h_x \end{bmatrix} \begin{bmatrix} I_n \\ C \end{bmatrix}.$$

The matrix C represents the response function that determines the current value of the response vector x in terms of the state vector s. The matrix P represents how the current state maps into the expectation of next period's state.

These functional equations may be solved explicitly using the QZ decomposition (Sims). They can be written as

$$Q \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix} \begin{bmatrix} Z_{11}^H & Z_{21}^H \\ Z_{12}^H & Z_{22}^H \end{bmatrix} \begin{bmatrix} P \\ CP \end{bmatrix} = Q \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} Z_{11}^H & Z_{21}^H \\ Z_{12}^H & Z_{22}^H \end{bmatrix} \begin{bmatrix} I_n \\ C \end{bmatrix},$$

[4] A log-linearized model is obtained using

$$\begin{bmatrix} I_n & 0_{mn} \\ -f_z h_{\dot{s}} & -f_z h_{\dot{x}} \end{bmatrix} D \begin{bmatrix} \Delta \ln(\dot{s}) \\ \Delta \ln(\dot{x}) \end{bmatrix} = \begin{bmatrix} g_s & g_x \\ f_s + f_z h_s & f_x + f_z h_x \end{bmatrix} D \begin{bmatrix} \Delta \ln(s) \\ \Delta \ln(x) \end{bmatrix}.$$

where

$$D = \begin{bmatrix} \mathrm{diag}(\bar{s}) & 0 \\ 0 & \mathrm{diag}(\bar{x}) \end{bmatrix}.$$

where Q and Z are unitary matrices and S and T are upper triangular. It is straightforward to verify that[5]

$$C = Z_{21}Z_{11}^{-1}$$

and

$$P = Z_{11}S_{11}^{-1}T_{11}Z_{11}^{-1}.$$

Letting $u = \text{diag}(S)$ and $v = \text{diag}(T)$, the ith eigenvalue of G is v_i/u_i. For the stability of the state process, it is necessary that these eigenvalues are less than 1 in absolute value. Stability can be ensured by a QZ decomposition that sorts in terms of the absolute value of the generalized eigenvalues (note that MATLAB's `qz` function does not). The MATLAB function `ress` and `relin` are available to solve for the steady state values and for the linearized equilibrium.

[5]Note that for unitary matrices

$$Z_{11}^H Z_{11} + Z_{21}^H Z_{21} = I$$

and

$$Z_{12}^H Z_{11} + Z_{22}^H Z_{21} = 0.$$

References

Fackler, Paul L. "A MATLAB Solver for Nonlinear Rational Expectation Models." *Computational Economics*, forthcoming.

Judd, Kenneth L. *Numerical Methods in Economics*. Cambridge, MA: MIT Press, 1998.

Marimon, Ramon and Andrew Scott, editors. *Computational Methods for the Study of Dynamic Economies*. Oxford: Oxford University Press, 1999.

Miranda, Mario J. and Paul L. Fackler. *Applied Computational Economics and Finance*. Cambridge MA: MIT Press, 2002.

Sims, Christopher A. "Solving Linear Rational Expectation Models." *Computational Economics* 20(2001):1–20.

Taylor, J.B. and H. Uhlig. "Solving Nonlinear Stochastic Growth Models: A Comparison of Alternative Solution Methods." *Journal of Business and Economic Statistics* 8(1990):1–18.