A General Approach to the Solution of Nonlinear Rational Expectations Models

Andrew P. Blake*
National Institute of Economic and Social Research
and
Richard G. Pierse
Department of Economics, University of Surrey

June 15, 2000

Abstract

A general method for the solution of rational expectations models is described. This focuses on the role of ordering rather than any particular algorithm, such as Gauss-Seidel or Newton, considering many approaches that have been previously proposed and showing where they fit into an overall general scheme. Each specific method is described by use of an algorithmic language built into WinSolve, a general nonlinear model solution program. Using this language a variety of methods can be easily compared.

Keywords: Nonlinear rational expectations models; solution

JEL classification: C50; C61; C63; C88

E-mail: ablake@niesr.ac.uk
Tel.: +44-20-7654-1924; Fax: +44-20-7654-1900

E-mail: r.pierse@surrey.ac.uk
Tel.: +44-1483-876953; Fax: +44-1483-303775

1 Introduction

In this paper a general approach to the solution of nonlinear rational expectations models is described which encompasses many previously proposed methods as special cases. This approach focuses on the role of ordering in model solution and shows how various solution methods can be interpreted as a natural outcome of adopting different orderings. Solution methods are

*Corresponding author: NIESR, 2 Dean Trench Street, London, SW1P 3HE, UK.

represented using an appropriate algorithmic language. This language provides a consistent framework for considering the various methods and has been implemented in WinSolve (Pierse (2000)), a Windows program for the solution of general economic models.

Early iterative solution methods were suggested by Anderson (1979) and Fair (1979), and first formalised by Fair and Taylor (1983). They adopted approaches which are simple extensions of the first-order solution methods (such as Gauss-Seidel) used for time-recursive models without rational expectations. Two key contributions by Stephen Hall (Hall (1985), Hall (1986)) refocused the literature. In the first he had the important insight that rational expectations solution should be interpreted as a 'stacked' system of $n \times T$ equations (where n is the number of equations and T the number of time periods). The resulting large sparse system (say of the order of 20 *thousand* equations) can be iterated over simultaneously to solution. He suggested that Gauss-Seidel could be applied to the whole system without regard to time structure. In the second paper Hall suggested that some equations might be better solved *backwards* rather than forwards if there is better information about values at the terminal date rather than the initial period. The real exchange rate is a perfect example of this, where the initial value will jump but the final value is usually known.

Becker and Rüstem (1993), Judd (1998).

Recent contributions have concentrated on the use of system-wide Newton methods, developing the ideas in Hall (1985). These rely on derivative information.[1] Both Laffargue (1990) and Armstrong, Black, Laxton, and Rose (1998) suggest alternative ways in which the sparsity of the system can be exploited to make feasible solving all the equations simultaneously.[2] Some comparative analysis between various methods has been carried out by Juillard, Laxton, McAdam, and Pioro (1998). In a recent survey McAdam

[1] Bischof, Carle, Corliss, Griewank, and Hovland (1992) have pioneered the use of a parsing program which reads computer code and supplies additional code for the model Jacobian. Alternatively, the derivatives can be evaluated by the compiler, at the same time as evaluating the expressions, a technique known as automatic differentiation (see Rall (1981)). WinSolve and other similar programs such as TROLL provide the derivatives through the compiler in this way.

[2] Well established coding for handling sparsity includes the Harwell MA48 Fortran library and the older MA28. See Duff, Erisman, and Reid (1986).

and Hughes Hallett (1999) compare available methods, noting that first order iterative schemes often have an advantage in much larger systems.[3]

Another way of exploiting sparsity is to adopt some form of block-based iteration. This has been widely used in practice in several different guises. It was common for multi-country models at least as far back as 1960s (see Duesenberry, Fromm, Klein, and Kuh (1969)) and continues in use, *e.g.* Faust and Tryon (1995). Individual blocks are solved one at a time and then iterated over to full solution. With only a few non-zero interactions between blocks this is an efficient strategy. Blocks are not necessarily limited to sets of equations in a single time period but may include groups of equations stacked over *all* time periods. An example of such a stacked-time block is the exchange rate system in Hall (1986). Solution of the individual blocks can be by any of the full system methods: first-order iterations (Gauss-Seidel or the like), Newton-type methods or even, say, Krylov methods. Furthermore, the Fair-Taylor approach can be viewed as a block-based approach, where the blocks are the equations in individual time periods and the connections between them are the future values.[4] Determining appropriate blocks for a model is an economic problem, related to the natural ordering of equations. The National Institute domestic model (NiDEM) for example uses nine separate blocks, each of which is determined by the characteristics of the model.

Often solution methods seem more diverse than they really are, and naive implementations of published algorithms are prone to failure. This paper is

[3]We should mention shooting methods, originally championed by Lipton, Poterba, Sachs, and Summers (1982), which exploit the two-point boundary-value structure of the problem. Equations with expectations are normalised on the maximum lead and solved forwards from arbitrary start values; initial conditions are then updated by some (Newton-type) rule and re-solved. Despite a variety of refinements this approach is infrequently used because models typically contain large unstable roots that cannot be solved forwards even over short horizons. Press, Teukolsky, Vetterling, and Flannery (1992) suggest that for the analogous differential equation problem a stacked-Newton approach is preferred.

[4]A successful variation on block-based approaches is to use incomplete iterations, proposed by Fisher, Holly, and Hughes Hallett (1986) and Fisher and Hughes Hallett (1988). Instead of iterating every individual block to full convergence, a partial solution is found, either by limiting the number of iterations or setting loose convergence criteria. This avoids wasting time achieving full convergence far from the final solution point. Although it may not help the convergence of models which would otherwise fail, this suggestion can often speed the iterative process considerably.

intended to to provide a bridge between published algorithms and practice. These have diverged as model users, expert in their respective models and software, have utilised 'tricks' that can be used to improve solution times and robustness. The approach described here is a formalization of the different suggestions by many authors over the past two decades. What our paper has in common with other published methods is that it presents the various iterative schemes algorithmically (although we choose to use an algorithmic language); what it has in common with practice is the role for heuristics. All technical details of underlying solution algorithms can be found in elementary numerical analysis textbooks, with Press, Teukolsky, Vetterling, and Flannery (1992) a particularly useful reference as it includes computer code. An up-to-date survey outlining how Gauss-Seidel, Newton and so on all work is McAdam and Hughes Hallett (1999).

The rest of the paper is organised as follows: Section 2 considers the ordering problem in rational expectations models which is illustrated graphically with a simple example. Section 3 describes a simple algorithmic language that is used in Section 4 to set out a variety of solution methods that have been proposed in the literature. It can be seen that all these methods are related by their treatment of ordering. Section 5 presents some conclusions.

2 Ordering rational expectations models

Equation ordering for models without rational expectations has mostly been associated with the approach of Don and Gallo (1987). Alternatively, Gilli (1992) and Gilli and Pauletto (1997) use graph theoretic methods to analyse the problem and suggest that ordering can be used to improve both the robustness and speed of model solution. There is an important role for ordering in methods that use either blocks or first-order iterations or both, but not for full system Newton methods where they make no difference. Recently, Hughes Hallett and Piscitelli (1998) have suggested that ordering can be used to speed model solution by some simple expedients. The approach to ordering outlined here is a similar method for speeding solution.

2.1 A simple model and ordering problem

The general solution problem is usually illustrated using a linear model. A linear constant coefficient model in first order form is written:

$$A_{\mathrm{C}} s_t = A_{\mathrm{L}} s_{t-1} + A_{\mathrm{F}} s^e_{t+1} + b_t \tag{1}$$

where the vector s_t of endogenous variables has as many 'dummy' leads and lags as required to make it into first-order form, and b_t is a vector of constants and initial and terminal conditions. In solving the model we assume internally consistent expectations so that $s^e_{t+1} = s_{t+1}$. Of course if $A_{\mathrm{F}} = 0$ then the model has no rational expectations of future variables.

The solution problem is to find s, the stacked values of s_t from $t = 1, \ldots, T$ such that:

$$\begin{bmatrix} A_{\mathrm{C}} & -A_{\mathrm{F}} & 0 & \cdots & 0 & 0 & 0 \\ -A_{\mathrm{L}} & A_{\mathrm{C}} & -A_{\mathrm{F}} & \cdots & 0 & 0 & 0 \\ 0 & -A_{\mathrm{L}} & A_{\mathrm{C}} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & A_{\mathrm{C}} & -A_{\mathrm{F}} & 0 \\ 0 & 0 & 0 & \cdots & -A_{\mathrm{L}} & A_{\mathrm{C}} & -A_{\mathrm{F}} \\ 0 & 0 & 0 & \cdots & 0 & -A_{\mathrm{L}} & A_{\mathrm{C}} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{T-2} \\ s_{T-1} \\ s_T \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{T-2} \\ b_{T-1} \\ b_T \end{bmatrix} \tag{2}$$

which is compactly written as:

$$As = b. \tag{3}$$

Any valid method can be used to solve the system simultaneously. These, of course, include the stacked-Newton variants, the Hall (1985) full system method or any other method which reduces the problem to a series of partial solutions which can be iterated on to complete convergence. A very important point to note is that here re-orderings of the matrix A are considered, which will enable a much richer set of solution methods to be described or implemented.

Later, a precise description of all the methods so far discussed will be given, using an algorithmic language. For now, we concentrate on the importance of ordering to our very general approach and begin by using a very concrete example which has enough features to illustrate the method.

Consider the following simple three equation model:

$$x_t = f(x^e_{t+1},\ y_{t-1}) \tag{4}$$

$$y_t = g(x_t,\ y_{t-1}) \tag{5}$$

$$z_t = h(y_t,\ z_{t-1}) \tag{6}$$

where $f(\cdot)$, $g(\cdot)$ and $h(\cdot)$ are (possibly linear) functions normalised on a particular left-hand side variable. Consider a five period problem, so (2) for this particular problem and one order of equations can be written:

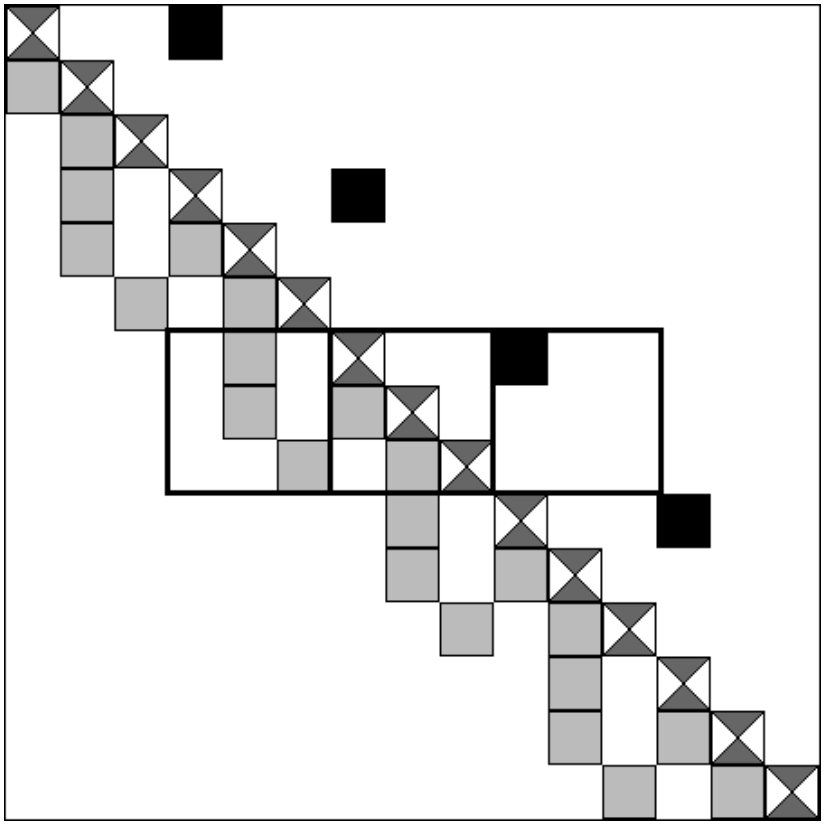
$$s = \left[\begin{array}{ccccccccccccccc} x_1 & y_1 & z_1 & x_2 & y_2 & z_2 & x_3 & y_3 & z_3 & x_4 & y_4 & z_4 & x_5 & y_5 & z_5 \end{array} \right]'.$$

This model can be considered essentially as linear, but here pictorial rather than parametric representation is used as this makes the point very clearly. The matrix A is represented by Figure 1. This is an *incidence matrix* where each equation is represented by a single row with an effect from another variable indicated by a non-white square. Note that in the middle a particular three 3×3 contiguous blocks are outlined. These represent from left to right the equivalents to the matrices $-A_{\mathrm{L}}$, A_{C} and $-A_{\mathrm{F}}$. Sufficient initial and terminal conditions for solution are assumed.

The differently shaded blocks have the following interpretations. Quartered boxes represent the normalised variable in equations (4)–(6). Black boxes represent a lead of the endogenous variable, or indeed anything that might appear in A_{F}. The light grey boxes indicate either contemporaneous or lagged values of any other variable that appear in a given equation, or lagged 'own' values.

This model has been chosen to have a number of salient features. Firstly, as ordered, it is recursive except for the forward expectations term. This would be untrue if, say, z were placed before y. Thus we have essentially 'sensibly' pre-ordered the model before considering the terms brought about by A_{F} being non-zero. In fact, this is rather implicit in any analysis of the rational expectations problem, where iterative methods based on partial solutions of some sort are often assuming at least some reasonable ordering for each sub-problem. Secondly, the Fair and Taylor (1983) and other methods can be illustrated by simple description, which is returned to below. Finally, it will very nicely show the properties of re-ordering schemes.

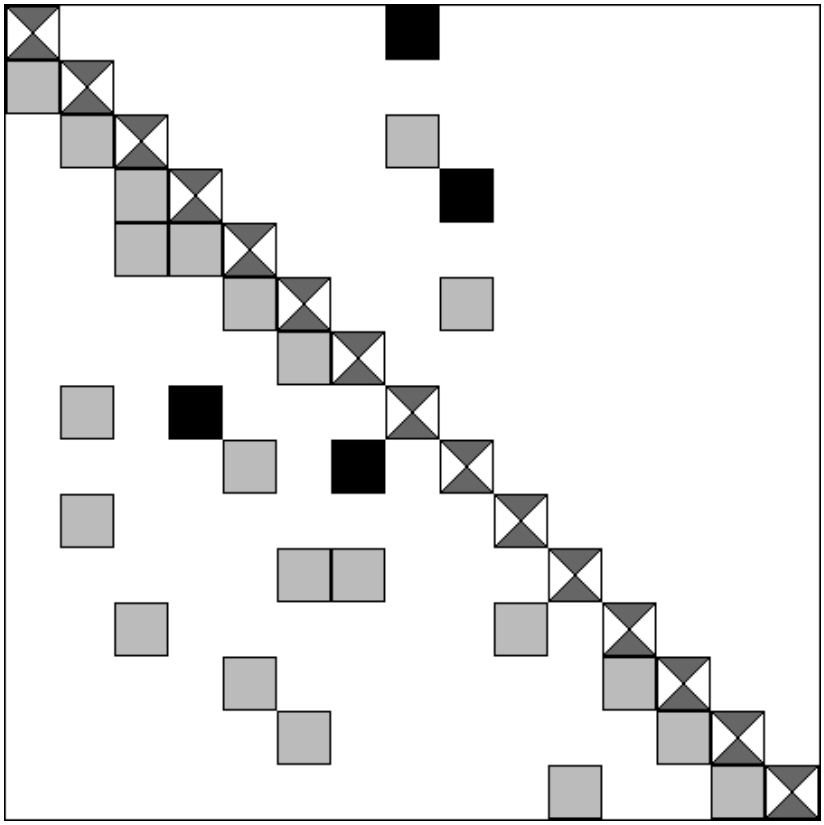
Figure 1: Unordered model



The Fair-Taylor method and variants then passes through the model ordered as in Figure 1 until convergence. For a more simultaneous model it would loop over the 3 equations until convergence is achieved (or not in the case of incomplete iterations). This is rather like separating the black squares out into a single block.

However, consider the equations simultaneously. An obvious (although not often used) strategy is the following. Perhaps a modeller should just reorder the whole system, find the best ordering on the basis of all equations simultaneously, and then simply go ahead and use full-system methods. Better ordering should help iterative methods, but they are irrelevant for Newton iterations, although modified Newton methods will be affected. To illustrate the outcome of this, the commonly used Don and Gallo (1987)

Figure 2: Don and Gallo (1987) ordered model



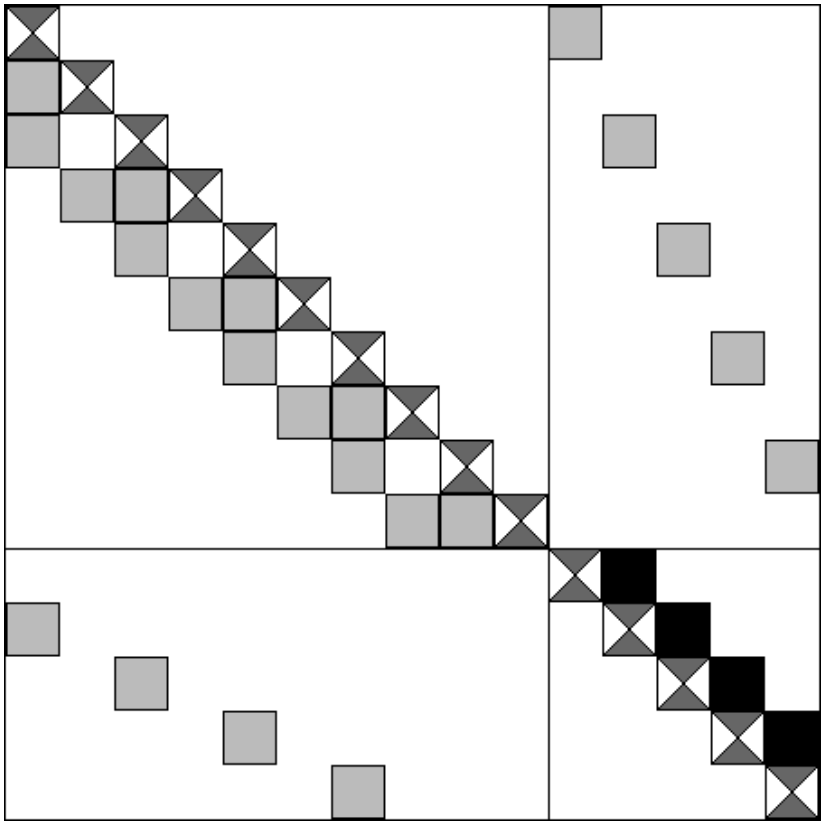
algorithm can be applied, to obtain the new ordering:

$$s = \left[\begin{array}{ccccccccccccccc} x_1 & y_1 & y_2 & x_3 & y_3 & y_4 & x_5 & x_2 & x_4 & z_1 & y_5 & z_2 & z_3 & z_4 & z_5 \end{array} \right]'$$

with the reordered A matrix illustrated in Figure 2.

Note that the z variables are now placed at the end, as they can easily be. But for the x variables this procedure gives a fascinating result. It is no longer the case that the leads (the black boxes) are all above the diagonal: Two appear below the diagonal. This is because the algorithm orders the x variable by first the odd period values and then the even. Therefore, the time structure of the model has been completely altered. Although this was done in a systematic way, it is difficult to rationalise with the economic structure.

Figure 3: Model re-ordered by variable and then time



However, we wish to be rather more systematic. Consider the following ordering:

$$s = \left[\begin{array}{ccccccccccccccc} y_1 & z_1 & y_2 & z_2 & y_3 & z_3 & y_4 & z_4 & y_5 & z_5 & x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \right]'$$

where the x variables are taken out from the main block and placed at the end. This is illustrated by an incidence matrix in Figure 3.

The pattern is emphasised by dividing the diagram into four. This ordering naturally suggests a block based iteration, where now there is a completely time-recursive block in the top left and a *backwards in time* recursive block in the bottom right. They depend on each other through the off-diagonal terms.

This is nothing other than the trick of taking the exchange rate out and solving it backwards. This therefore demonstrates the good sense in the

approach suggested by Hall (1986), but motivates it by ordering.

2.2 A synthesis of existing approaches

Figure 3 contains the key to the approach we suggest. Some variables are more naturally affected by their own leads and lags than any other simultaneous variable. These are more properly iterated over first *through time* and *then* across equations, as updating next (or last) periods value has more impact than updating the concurrent value of another. Others are very naturally affected in a nearly (or completely) time-recursive way once other variables are treated as a separate block. These are properly iterated across equations and then through time.

This suggests the following strategy. Take as one group variables which are very forward-looking or depend closely on such variables: nominal variables such wages and prices might be an example. Take as another group variables which do not have such dependencies: many real variables are like this. Having separated these out, there are likely to be a large number of pre- and post-recursive variables to those blocks that can be ordered. There are often several distinct groups which can be naturally separated out. In the National Institute model, NiDEM, there are three groups linked by other recursive blocks.

Each block is then solved sequentially, taking into account dependencies (the off-diagonal elements in Figure 3). The variables in the blocks themselves are determined by what drives Figure 3: namely ordering. Each block may be solved iteratively forwards or backwards, or by Newton per period or stacked-Newton for every period but for a subset of variables. Each iterative scheme can also be incomplete as long as a full convergence check is done at the end.

This scheme is best understood by example. We show how methods inter-relate in the next section.

3 An Algorithmic Description Language

To describe the algorithms a formal language is used. This is a simplified version of the algorithmic language developed in WinSolve to specify user-

defined solution algorithms or 'DIY' files.[5] Commands and keywords in this language are case insensitive and keywords can appear in any order.

The following conventions are used:

bold	indicates a command or keyword that should be typed as shown
italic	indicates a parameter or value to be supplied by the user
[ ]	denotes an optional keyword or parameter which may be omitted
...	(ellipsis) denotes input is a series of values as indicated
bname	denotes a name to identify a block of equations
lname	denotes a name to identify a loop
vname	denotes the name of a variable in the current model

3.1 Commands

Commands are organised into three groups. These control the equation block definitions, solution of the blocks of equations and the looping structure itself. As the language is recursive, loops can be nested to any depth and quite arbitrary schemes can be implemented.

3.1.1 Equation block definition

The **eqn** command defines the equations that comprise a given block:

eqn *bname vname1 ... vnamek* ;

where block *bname* is defined to consist of equations for variables *vname1* to *vnamek*. These equations will be solved in the order that they are listed. Equations may appear in more than one block.

3.1.2 Solution control

Equation blocks are solved using the following command:

solve *bname* [**damp** *f*]

This solves the block of equations *bname* with **damp** the solution damping factor to be used for this equation block. Setting a damping factor will

[5] The acronym DIY stands for 'Do It Yourself', which seems appropriate for something user-defined.

override the globally set default, and can therefore also be used to remove damping from a block.

3.1.3 Looping control

Three different types of loops are supported. The first is the basic:

loop *lname* [**alg** *type*] [**itmax** *n*] [**abs** *f*] [**pct** *g*]

which starts an iteration loop named *lname*. It is terminated by matching '**end** *lname*' command. The argument **alg** determines the solution method for the loop which can be one of Jacobi, Gauss-Seidel (GS, the default), Fast Gauss-Seidel (FGS) or Newton. The maximum number of iterations is controlled by **itmax** and the two (absolute and percentage) convergence criteria by **abs** and **pct**.

This command is used for controlling the number of times any of the other commands is repeated, the termination of the loop being determined by the convergence criteria. This command will generally be used in conjunction with time looping, which is achieved by either

forw *lname*

which starts a time loop with *lname* running forwards in time and is terminated by matching '**end** *lname*' command or

backw *lname*

which starts a time loop named *lname* running backwards in time. Again, it is terminated by matching '**end** *lname*' command. Note that loops *must* be nested correctly.

The ability to make up loops using these commands for different blocks of equations is the key to developing general solution methods using theory-based iterations. Examples of these are given below.

3.2 Terminal conditions

The command **term** with no arguments updates the terminal conditions independent of their position relative to equation block definitions. This

is a useful way of replicating other methods, although is rather against the overall approach outlined. It can be used as an 'all at once' call to ensure that the most up-to-date information is used. WinSolve can instead be operated so that the terminal conditions are updated in a 'just-in-time' fashion, where the special equation for data past the end is called when the required data would otherwise be missing. In the current National Institute model of the UK economy, NiDEM, the current treatment is that terminal conditions are coded up explicitly for calling in the 'just-in-time' fashion. This is a convention rather than an option and is an important part of what defines that solution method. In earlier incarnations of NiDEM, terminal conditions were only updated by (equivalent) calls to **term**.

The updating of terminal conditions can make a substantial difference to solution. Simply updating the terminal condition at some time during the inner iteration can be markedly different to the 'just-in-time' approach.

4 Solution algorithms as DIY files

WinSolve has several built-in solution methods in addition to DIY files. These include 'canned' versions of the Fair and Taylor (1983) and Laffargue (1990). The available options for Fair and Taylor (1983) allow for Newton inner iterations (as in Juillard, Laxton, McAdam, and Pioro (1998)) and incomplete iterations (as in Fisher and Hughes Hallett (1988) and as a limiting case Hall (1985)). These flexible implementations allow the user many choices in, say, the treatment of terminal conditions or choice of Jacobi, Gauss-Seidel, Fast-Gauss-Seidel or Newton iterative algorithms.

However, using the DIY files it is possible to implement *exactly as written* all the methods outlined above using the DIY language and many possible variations. In particular, we can apply the method used by the National Institute of Economic and Social Research to solve their UK model, NiDEM 2000. This method is a synthesis of several previous approaches, relying on an analysis of 'whole-model' ordering.

In this section we give examples of DIY files that reproduce the Fair-Taylor, Hall and Fisher-Hughes Hallett methods for a simple model. Fair-Taylor is taken as the benchmark method and we indicate where the others

depart from it. The example model is purely illustrative but contains features that can be exploited more generally.

The twelve 'dummy' equations are initially split into five groups to replicate the types of equation blocks commonly found. *armas* contains equations that do not depend on any endogenous variables and can be solved before the outer iteration loop. This typically consists of auto-regressive equations. *prerecs* are pre-recursive equations and so, in each time period, can be solved before the inner iteration loop. *simul* is the main simultaneous block. *exch* is a block of equations that have some effect on variables in the simultaneous block. The equations in *postrecs* are post-recursive and so, in each time period can be solved after the inner iteration loop. Finally, *outputs* contains equations that do not feed back into the other equations at all and so can be solved after the outer iteration loop. It is the triple *prerecs*, *simul* and *postrecs* which will receive most attention in the generalised method.

The global damping factor must be set to unity as without it *armas* and *outputs* will not solve correctly. The equations in the *simul* block are solved with a damping factor of 0·7. The convention is followed that **eqn** statements are grouped together at the end, and that indentation is used to aid readability. To the left, the lines are numbered. This is particularly useful as some different methods require only minor modification of individual lines.

4.1 The Fair and Taylor (1983) method

Here we outline how to recreate the Fair and Taylor (1983) solution using a DIY file. We create so-called Type I (inner loop) and Type II (outer loop) iterations, as the need for Type III iterations is usually obviated by the use of appropriate terminal conditions and experimentation with the terminal date. The algorithm is shown in Table 1 and the equation definitions are included in Table 2.

We omit the equation block definitions for convenience but retain the same numbering. It is an important feature of the overall approach outlined here that there is a close connection between the equation block definitions and the 'algorithm'. When considering other algorithms, additional equation blocks will need to be defined.

```
1.    WinSolve diy file
2.    forw armaloop
3.          solve armas
4.    end armaloop

5.    loop outerloop itmax 300 pct .0001
6.          term
7.          forw mainloop
8.                solve prerecs
9.                loop innerloop
10.                     solve simul damp 0.7
11.                     solve exch
12.               end innerloop
13.               solve postrecs
14.         end mainloop
15.   end outerloop

16.   forw outloop
17.         solve outputs
18.   end outloop
```

Table 1: Fair and Taylor (1983) algorithm.

19. **eqn** *armas*
20. Y10 Y2;
21. **eqn** *prerecs*
22. Y5 Y4;
23. **eqn** *simul*
24. Y3 Y6 Y8 Y11;
25. **eqn** *exch*
26. Y7;
27. **eqn** *postrecs*
28. Y1;
29. **eqn** *outputs*
30. Y9 Y12;

Table 2: Equation definitions (Version 1).

Solution relies on the loop *outerloop* which is simply a check that all variables have converged. The **forw** loop *mainloop* iterates sequentially through time with the solution over all equations which are fully solved out by *innerloop*.

There is a question as to where is the best place to insert the call to **term**, to update the terminal conditions. It could be in the **forw** loop instead. This makes a number of redundant calls to **term** (*i.e.* where the forecast horizon in individual equations is less than the number of periods-to-go in the loop), but it mimics the just-in-time approach. In a number of Fair-Taylor implementations we have seen, the equivalent calls to **term** are as in Table 1. We prefer the just-in-time approach, and therefore would remove the call to **term** and set just-in-time terminal conditions as a global option.

4.2 The Stacked-Newton approach

The algorithmic language allows us to recreate the systems approach of Laffargue (1990). With such a system approach, a 'just-in-time' terminal condition becomes an extra equation to be determined simultaneously. In Table 3, although the pre-recursive and post-recursive variables have been

```
1.    WinSolve diy file
2.    forw armaloop
3.          solve armas
4.    end armaloop
5.    loop newtonloop alg newton itmax 30 pct .0001
6.          solve prerecs
7.          solve simul
8.          solve exch
9.          solve postrecs
10.   end newtonloop
11.   forw outloop
12.         solve outputs
13.   end outloop
```

Table 3: The Stacked-Newton algorithm.

separated out,[6] all three blocks are looped over together. The number of iterations is set to 30: This is many less than the Gauss-Seidel type loops in Table 1, but convergence of derivative methods will normally be faster. The block structure is irrelevant except for the variables contained in *armas* and *outputs*.

4.3 The Fisher and Hughes Hallett (1988) method

Fisher and Hughes Hallett (1988) noted that the solution of inner loops to full convergence can be extremely wasteful when far from a solution. They suggested that partial solution, dictated either by a maximum number of inner iterations or a loose convergence criteria (or both), can substantially reduce the number of iterations required in the early part of the solution without necessarily increasing the number of outer iterations. In the DIY file this is achieved by judicious choice of the **itmax** or **pct** parameters in the inner loop **solve** command (line 9 in Table 1). This turns out to be a very effective strategy for speeding up solution, and incomplete iterations between blocks should *always* be experimented with for any block-based scheme. This is probably the most important practical advance in nonlinear

[6]The WinSolve implementation does this automatically.

```
5.    loop outerloop itmax 300 pct .0001
6.         term
7.         forw mainloop
8.              solve prerecs
9.              loop innerloop itmax 10 pct .01
10.                  solve simul damp 0.7
11.                  solve exch
12.             end innerloop
13.             solve postrecs
14.        end mainloop
15.   end outerloop
```

Table 4: Fisher and Hughes Hallett (1988) algorithm.

RE model solution.

4.4 The Hall (1985) method

As noted above, Hall (1985) suggested that the Fair and Taylor (1983) method did not take sufficient account of the simultaneous structure of the rational expectations solution problem. Looping vertically through the equations stacked by time period simply treats the whole system as a single set of dynamic equations to be solved subject to initial and terminal conditions. It is this simultaneous approach that makes the so-called 'stacked-Newton' methods attractive.

This method requires very little modification to the solution procedure suggested above. Indeed, in the WinSolve DIY file, only one line needs to be altered, and then only by adding an additional option. In line 9 of Table 1 we could impose a maximum iteration limit of 1. Note that this performs the stacking procedure implicitly given the equation ordering in the equation blocks.

In Table 5 we show the relevant part of the code which has been simplified by simply removing the loop *innerloop*.

```
5.    loop outerloop itmax 300 pct .0001
6.         term
7.         forw mainloop
8.              solve prerecs
9.              solve simul damp 0.7
10.             solve exch
11.             solve postrecs
12.        end mainloop
13.   end outerloop
```

Table 5: Hall (1985) algorithm.

4.5 Hall (1986) method

What if we had identified a variable that could be naturally solved backwards? Hall suggested that when the information about a variable is its terminal condition rather than its initial condition, then it makes sense to solve that variable backwards in time. In our model, the equation block *exch* has been included to identify such variables which will be solved in a separate loop. Table 6 includes the **backw** loop. In our example, there is only a single equation in the block so that it does not need to be solved iteratively and the loop *exchloop* is strictly redundant. More generally, iteration over the block would be necessary. Iteration is over time before equations, just as would be appropriate for Figure 3. The method can easily be generalised to allow for incomplete iterations between blocks of equations.

4.6 Block-stacked-Newton algorithm

Blocks can equally well be solved by Newton rather than first-order methods.Table 7 shows an inner loop solved by Newton and the *exch* equation block is solved by stacked-Newton, all periods at once.

The advantage here is that the very forward-looking variables can be solved using stacked-Newton but the others by cheaper first-order methods. This reduces the size of the system considerably and, if iteration between blocks is cheap, then this might be an exceptionally cheap scheme.

```
5.    loop outerloop itmax 300 pct .0001
6.         term
7.         forw mainloop
8.              solve prerecs
9.              loop innerloop
10.                  solve simul damp 0.7
11.             end innerloop
12.             solve postrecs
13.        end mainloop
14.        loop exchloop
15.             backw backloop
16.                  solve exch
17.             end backloop
18.        end exchloop
19.   end outerloop
```

Table 6: Hall (1986) algorithm.

```
5.    loop outerloop itmax 100 pct .0001
6.         forw mainloop
7.              solve prerecs
8.              loop innerloop alg newton
9.                   solve simul
10.             end innerloop
11.             solve postrecs
12.        end mainloop
13.        loop exchloop alg newton
14.             solve exch
15.        end exchloop
16.   end outerloop
```

Table 7: A block-stacked-Newton algorithm.

4.7 The NIESR NiDEM 2000 solution scheme

Finally, we sketch the solution scheme used for the National Institute domestic model NiDEM 2000. Table 9 shows the blocks used. The main blocks are: MODEL (mainly real variables), REVREX (prices, wages and the exchange rate) and REVEQ (financial variables and equity prices). These are linked by a series of other blocks. There is also a London submodel.

Table 8 illustrates the DIY file. It demonstrates a mixture of blocks solved over time before across equations (REVREX and REVEQ) and in the standard way (MODEL), with incomplete iteration and varying damping factors. This sophisticated scheme relies on a model ordered by choice of variables in each block. It is a very robust solution scheme, reflecting considerable experience with the model.

```
1.  WinSolve diy file
2.  forw armaloop
3.      solve ARMAS damp 1
4.  end armaloop
5.  loop outer itmax 500
6.      forw modeltime
7.          solve PREREC
8.          loop modelloop itmax 10
9.              solve MODEL
10.         end modelloop
11.         solve POSMOD
12.     end modeltime
13.     loop pwexloop itmax 500
14.         forw pwextime
15.             solve REVREX
16.         end pwextime
17.     end pwexloop
18.     forw postpw
19.         solve POSPW
20.     end postpw
21.     loop eqloop itmax 10
22.         forw eqtime
23.             solve REVEQ
24.         end eqtime
25.     end eqloop
26. end outer
27. forw outtime
28.     solve POSOUT damp 1
29. end outtime
30. forw lontime
31.     loop lonloop
32.         solve LONDON
33.     end lonloop
34. end lontime
```

Table 8: NIESR NiDEM 2000 forecasting model.

Block name	No. of variables	Any damping
ARMAS	56	No
PREREC	20	No
MODEL	156	Yes
POSMOD	13	No
REVREX	126	Yes
POSPW	49	No
REVEQ	41	Yes
POSOUT	24	No
LONDON	65	No

Table 9: Block definitions

5 Conclusions

In this paper, we have stressed the importance of the ordering problem as a way of viewing different solution algorithms and have developed a framework in which existing algorithms can be described and new ones developed.[7] In practice, a large number of models, including NiDEM and models at the Bank of England and the London Business school, are solved using an eclectic approach involving an amalgam of the standard algorithms, taking into account knowledge of the economic structure of the model. These methods have in general proved very successful although they are rarely documented and are often hardcoded into software, making them difficult to replicate and evaluate. The algorithmic language described here represents a step towards making the algorithms more transparent.

The knowledge required to implement the sort of solution procedure outlined in Table 8 is simply a matter of the relevant economics. The blocks are determined, for example, by whether the variables are forward looking and closely related or not. In reality the very organisation process that goes along with building a model is likely to reveal such orderings to the modeller.

Finally, WinSolve provides a convenient framework for analysing solution methods. McAdam and Hughes Hallett (1999) suggest how comparisons between methods should be made and in particular stress the importance of

[7] We are currently investigating the usefulness of partial stacked Newton solution in NiDEM.

a common modelling framework. Given that models are often solved using tricks known only to the modeller, the WinSolve provided DIY files allows exact comparisons to be made.

References

Anderson, P. (1979): "Rational Expectations Forecasts from Non-Rational Models," *Journal of Monetary Economics*, 67(1), 101–115.

Armstrong, J., R. Black, D. Laxton, and D. Rose (1998): "A Robust Method for Simulating Forward-Looking Models," *Journal of Economic Dynamics and Control*, 22(4), 489–501.

Becker, R., and B. Rüstem (1993): "Algorithms for Solving Nonlinear Dynamic Decision Models," *Annals of Operations Research*, 44, 117–142.

Bischof, C., A. Carle, G. Corliss, A. Griewank, and P. Hovland (1992): "ADIFOR: Generating Derivative Codes from Fortran Programs," *Scientific Programming*, 1(1), 11–29.

Don, F., and G. Gallo (1987): "Solving Large Sparse Systems of Equations in Econometric Models," *Journal of Forecasting*, 6, 167–180.

Duesenberry, J., G. Fromm, L. Klein, and E. Kuh (1969): *The Brookings Model: Some Further Results*. North-Holland Publishing Company, Amsterdam.

Duff, I., A. Erisman, and J. Reid (1986): *Direct Methods for Sparse Matrices*. Oxford University Press.

Fair, R., and J. Taylor (1983): "Solution and Maximum Likelihood Estimation of Dynamic Rational Expectations Models," *Econometrica*, 51(4), 1169–1185.

Fair, R. C. (1979): "An Analysis of a Macro-Econometric Model with Rational Expectations in the Bond and Stock Markets," *American Economic Review*, 69(4), 539–552.

Faust, J., and R. Tryon (1995): "A Distributed Block Approach to Solving Near-Block-Diagonal Systems with an Application to a Large Macroeconometric Model," *Computational Economics*, 8(4), 303–316.

Fisher, P., S. Holly, and A. Hughes Hallett (1986): "Efficient Solution Techniques for Nonlinear Rational Expectations Models," *Journal of Economic Dynamics and Control*, 10(1/2), 139–145.

Fisher, P., and A. Hughes Hallett (1988): "Efficient Solution Techniques for Linear and Nonlinear Rational Expectations Models," *Journal of Economic Dynamics and Control*, 12(4), 635–657.

Gilli, M. (1992): "Causal Ordering and Beyond," *International Economic Review*, 33(4), 957–971.

Gilli, M., and G. Pauletto (1997): "Sparse Direct Methods for Model Simulation," *Journal of Economic Dynamics and Control*, 21(6), 1093–1111.

Hall, S. (1985): "On the Solution of Large Economic Models with Consistent Expectations," *Bulletin of Economic Research*, 37(2), 157–161.

——— (1986): "Time Inconsistency and Optimal Policy Formulation in the Presence of Rational Expectations," *Journal of Economic Dynamics and Control*, 10(1/2), 323–326.

Hughes Hallett, A., and L. Piscitelli (1998): "Simple Reordering Techniques for Expanding the Convergence Radius of First-Order Iterative Techniques," *Journal of Economic Dynamics and Control*, 22(8/9), 1319–1333.

Judd, K. L. (1998): *Numerical Methods in Economics*. The MIT Press.

Juillard, M., D. Laxton, P. McAdam, and H. Pioro (1998): "An Algorithm Competition: First-Order Iterations versus Newton-Based Techniques," *Journal of Economic Dynamics and Control*, 22(8/9), 1291–1318.

Laffargue, J.-P. (1990): "Résolution d'un Modèle Macroéconomique avec Anticipations Rationnelles," *Annales D'Economie et de Statistique*, 17(Janvier/Mars), 97–119.

Lipton, D., J. Poterba, J. Sachs, and L. Summers (1982): "Multiple Shooting in Rational Expectations Models," *Econometrica*, 50(5), 1329–1333.

McAdam, P., and A. Hughes Hallett (1999): "Nonlinearity, Computational Complexity and Macroeconomic Modelling," *Journal of Economic Surveys*, 13(5), 577–618.

Pierse, R. G. (2000): *WinSolve Version 3.0*. Program and documentation available from `http://www.econ.surrey.ac.uk/winsolve/`.

Press, W., S. Teukolsky, W. Vetterling, and B. Flannery (1992): *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edn.

RALL, L. (1981): *Automatic Differentiation: Techniques and Applications.* Springer Verlag, *Lecture Notes in Computer Science*, Volume 120.