OPTIMAL CONTROL IN NONLINEAR MODELS: A GENERALISED GAUSS-NEWTON ALGORITHM WITH ANALYTIC DERIVATIVES

By

Richard G. Pierse
(University of Surrey)

DP 09/06

Department of Economics
University of Surrey
Guildford
Surrey GU2 7XH, UK
Telephone +44 (0)1483 689380
Facsimile +44 (0)1483 689548
Web www.econ.surrey.ac.uk
ISSN: 1749-5075

Optimal control in nonlinear models: a generalised Gauss-Newton algorithm with analytic derivatives

Richard G. Pierse
Department of Economics
University of Surrey, Guildford GU2 7XH, U.K.

February 24 2006

Abstract

In this paper we propose an algorithm for the solution of optimal control problems with nonlinear models based on a generalised Gauss-Newton algorithm but making use of analytic model derivatives. The method is implemented in WinSolve, a general nonlinear model solution program.

1 Introduction

In recent years, Newton methods have largely replaced first-order techniques as the primary solution method for non-linear deterministic economic models. As argued in Juillard *et al.* (1998), Newton techniques are faster and more robust than first-order techniques such as those based on Gauss-Seidel iterations and, as is well known, are quadratically rather than just linearly convergent. Newton methods require the evaluation of first order model derivatives. In models without any forward-looking behaviour by agents, the model equations can be solved recursively, period-by-period, requiring, in each period from 1 to T, the solution of an $n \times n$ linear system of first-order derivatives (n being the number of current-dated endogenous variables). When the model contains forward-looking variables, the system of equations is no longer recursive and so must be stacked into a simultaneous system of nT equations. This system can be very large and solving it efficiently requires use of sparse solution techniques. The most widely used algorithm, due to Laffargue (1990), Boucekkine (1995) and Julliard (1996) and known as the L-B-J algorithm, makes use of the special block-band structure of the Jacobian matrix to solve the equations efficiently by Gaussian block pivoting on matrices of order $n \times n$. The model derivatives required for Newton methods can be computed numerically using finite-difference numerical techniques but more often automatic derivatives are used. Automatic derivative techniques (see Rall (1981) or Griewank (2000)) apply analytic derivative formulae to compute first (and higher order) derivatives as a by-product of evaluation of the (parsed) model equations. Automatic derivatives are generally much cheaper to compute than numeric derivatives and are more accurate. Computer packages such as *Dynare* (Juillard, 1996), *Troll* (Hollinger, 1996) and *WinSolve* (Pierse, 2002) all implement stacked-Newton as the default algorithm for solving deterministic forward-looking models. In the current version of *Dynare*, the first order model derivatives are calculated numerically but both *Troll* and *WinSolve* implement automatic derivatives.

Given the prevalent use of Newton methods to solve economic models, it is also natural to consider them in the solution of optimal control problems with an economic model. A control problem involves the minimisation of a (typically quadratic) loss function subject to the equations of the model. The optimiser is assumed to be able to choose values for a set of (exogenous) control variables in order to achieve the targets defined by the loss function. The first order conditions for the control problem involve the derivatives of the loss function with respect to the control variables. These first order conditions can be found by solving a set of nT linear equations involving the model Jacobian matrix stacked over time. When the model has no forward-looking variables, it may be possible to solve this problem recursively, period by period, as long as the matrix of derivatives of current period target variables with respect to current period control variables is square and non-singular. However, when the model contains forward-looking variables or the control variables only affect the target variables with a lag, solving the full system of nT linear equations is required. Given the size of this system, solving the problem efficiently requires the use of sparse solution techniques.

In this paper, an algorithm is developed utilising the block-band structure of the Jacobian matrix to solve the first order equations of the control problem efficiently by Gaussian block-pivoting in a way similar to that used in the L-B-J algorithm. This method is then applied in an implementation of the Gauss-Newton solution method using analytic derivatives computed using techniques of automatic differentiation. This algorithm has been implemented in the current version of *WinSolve*, a user-friendly computer package for solving general nonlinear models, described in Pierse (2002)[1].

The structure of the paper is as follows: Section 2 briefly describes the stacked-Newton solution method and the L-B-J algorithm. Section 3 discusses the (generalised) Gauss-Newton algorithm for solving non-linear quadratic minimisation problems and develops an algorithm for computing the first order conditions of the problem. Finally, some conclusions are drawn.

2 Stacked Newton and the L-B-J algorithm

Consider solution of the general nonlinear set of deterministic equations

$$\mathbf{f}(\mathbf{y}_t, \mathbf{y}_{t+1}, \cdots, \mathbf{y}_{t+q}, \mathbf{y}_{t-1}, \cdots, \mathbf{y}_{t-p}, \mathbf{x}_t; \boldsymbol{\theta}) = \mathbf{0}\,, \quad t = 1, \cdots, T \tag{1}$$

where $\mathbf{y}_t$ is an $n \times 1$ vector of *endogenous* variables in time period t, $\mathbf{x}_t$ is an $m \times 1$ vector of current and lagged exogenous variables, $\mathbf{f}$ is an $n \times 1$ vector valued function and $\boldsymbol{\theta}$ is a vector of parameters, p is the longest lag in the model and q is the longest lead. This system represents a set of n nonlinear equations over T time periods. Stacking the equations over all time periods produces a set of nT equations. The Jacobian matrix of this stacked system has a special structure and looks like

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & \mathbf{F}_1^1 & \cdots & \mathbf{F}_1^q & & & & \\ \mathbf{B}_2^1 & \mathbf{J}_2 & \mathbf{F}_2^1 & \cdots & \mathbf{F}_2^q & & & \\ \vdots & \ddots & \ddots & \ddots & \cdots & \ddots & & \\ \mathbf{B}_p^p & \cdots & \mathbf{B}_p^1 & \mathbf{J}_p & \ddots & \cdots & \ddots & \\ & \ddots & \cdots & \ddots & \ddots & \ddots & \cdots & \mathbf{F}_{T-k}^q \\ & & \ddots & \cdots & \ddots & \ddots & \ddots & \vdots \\ & & & \ddots & \cdots & \ddots & \mathbf{J}_{T-1} & \mathbf{F}_{T-1}^1 \\ & & & & \mathbf{B}_T^p & \cdots & \mathbf{B}_T^1 & \mathbf{J}_T \end{bmatrix} \tag{2}$$

[1]A free trial version is available for download from the Internet at web address www.econ.surrey.ac.uk/winsolve.

where

$$\mathbf{J}_t = \frac{\partial \mathbf{f}}{\partial \mathbf{y}'_t}, \quad \mathbf{F}^i_t = \frac{\partial \mathbf{f}}{\partial \mathbf{y}'_{t+i}}, \quad \mathbf{B}^i_t = \frac{\partial \mathbf{f}}{\partial \mathbf{y}'_{t-i}}$$

are all matrices of dimension $n \times n$.

The *Stacked Newton method* applies Newton's method (Newton (1686)) to the stacked system. This involves iterating on the set of nT equations

$$\mathbf{J}(\mathbf{y}^s - \mathbf{y}^{s-1}) = -\mathbf{f}(\mathbf{y}^{s-1}) \tag{3}$$

where $\mathbf{y}^s$ is the $nT \times 1$ vector of stacked values of the endogenous variables in iteration s. Iterations start from an initial guess at the solution, $\mathbf{y}^0$, and terminate when a convergence criterion such as

$$\max_j \left| \frac{\mathbf{y}^s_j - \mathbf{y}^{s-1}_j}{\mathbf{y}^{s-1}_j} \right| < \varepsilon$$

has been satisfied, for some small value of ε.

Each iteration of Newton's method involves the solution of a set of nT equations and, when either n or T is big, the $nT \times nT$ Jacobian matrix $\mathbf{J}$ will be very large. Since the cost of solution of a set of k linear equations, using standard techniques such as the LU decomposition, is roughly of order $O(k^3)$, (see Duff *et al.* (1986) or Judd (1998)), this cost can quickly become prohibitive. Even the storage of the full Jacobian matrix in computer memory can be a problem: for example with $nT = 5000$, 182 megabytes of memory is required, and with $nT = 10000$, 728 megabytes.

It can be seen from (2) that the structure of the Jacobian matrix is very sparse with many zero elements. A solution to both the storage and computational problems is to take account of the known sparsity of the Jacobian matrix. Two approaches are possible. One approach, suggested by Armstrong *et al.* (1998) is to make use of general sparse matrix solution methods as described in Duff *et al.* (1986) and implemented by AERE Harwell in the MA28 library (Duff (1977))[2]. These methods use the known sparsity pattern of a matrix to reduce the required storage by storing only non-zero elements, and to eliminate redundant calculations involving zero elements.

An alternative approach is the L-B-J algorithm, originally suggested by Laffargue (1990) and refined by Boucekkine (1995) and Julliard (1996). This approach explicitly takes account of the special block-band structure of the

[2]The NAG library also includes similar versions of these routines.

Jacobian matrix (2) to solve the equations efficiently by Gaussian block pivoting on matrices of dimension $n \times n$. The method proceeds in two stages. In the first stage, the Jacobian matrix is transformed into an upper block-triangular structure, by eliminating the blocks below the diagonal by the recursion

$$\text{subtract } \mathbf{B}_t^j \times \text{rows of block } t-j \text{ from rows of block } t$$

from $j = p^*, p^*-1, \cdots, 1$, where $p^* = \min(p, t-1)$ and then replacing the block on the diagonal by the identity matrix by the operation

$$\text{premultiply rows of block } t \text{ by the inverse of the diagonal block } \mathbf{J}_t^*$$

where $\mathbf{J}_t^*$ is the diagonal block $\mathbf{J}_t$ after transformation by the set of Gaussian eliminations.

The algorithm proceeds, period by period from $t = 1$ through to $t = T$. Note that the only blocks that need to be stored are those corresponding to the lead coefficients $\mathbf{F}_t^i$, $i = 1, \cdots, q$. This means that storage is reduced from $nT \times nT$ to $nT \times nq$. This can be reduced further by dropping any columns in $\mathbf{F}_t^i$, corresponding to variables that never appear with a lead. Note also that the last step in this stage is the solution of an $n \times n$ system of equations in the transformed Jacobian block $\mathbf{J}_t^*$. Since this matrix will itself usually be sparse, the general sparse solution methods of Duff *et al.* (1986) can be applied to this step. (This combination of general sparse matrix techniques with the L-B-J algorithm is used in the implementations of the algorithm in the computer packages *Troll* and *WinSolve*.)

Finally, in the second stage of the procedure, the upper block-triangular structure is solved recursively, going backwards in time from period T to period 1.

3 Gauss-Newton and analytic derivatives

Consider solution of the nonlinear quadratic optimisation problem

$$\min_{\mathbf{u}} L = \frac{1}{2}(\mathbf{y}-\mathbf{y}^*)'\mathbf{W}(\mathbf{y}-\mathbf{y}^*) + \frac{1}{2}(\mathbf{u}-\mathbf{u}^*)'\mathbf{Q}(\mathbf{u}-\mathbf{u}^*) \tag{4}$$

subject to the set of non-linear model equations

$$\mathbf{f}(\mathbf{y}_t, \mathbf{y}_{t+1}, \cdots, \mathbf{y}_{t+q}, \mathbf{y}_{t-1}, \cdots, \mathbf{y}_{t-p}, \mathbf{u}_t, \mathbf{z}_t; \boldsymbol{\theta}) = \mathbf{0}\,, \quad t = 1, \cdots, T. \tag{5}$$

Here the model is as before except that the exogenous variables $\mathbf{x}_t$ have now been separated into k control variables $\mathbf{u}_t$ and $(m-k)$ other exogenous variables $\mathbf{z}_t$. In the quadratic loss function (4) $\mathbf{y}^*$ and $\mathbf{u}^*$ represent desired values of the target variables and controls respectively and $\mathbf{W}$ and $\mathbf{Q}$ represent positive semi-definite block diagonal matrices of weights of dimension $nT \times nT$ and $kT \times kT$ respectively.

The Gauss-Newton algorithm is an iterative procedure based on a second order Taylor approximation to the original problem in which the Hessian matrix

$$\frac{\partial^2 L}{\partial \mathbf{u} \partial \mathbf{u}'}$$

is approximated by the expression

$$\frac{\partial^2 L}{\partial \mathbf{u} \partial \mathbf{u}'} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}}' \mathbf{W} \frac{\partial \mathbf{y}}{\partial \mathbf{u}} + \mathbf{Q} \tag{6}$$

which drops all terms involving second order model derivatives. The rationale for this is that, in the neighbourhood of a solution when $\mathbf{y}$ and $\mathbf{u}$ are close to their targets of $\mathbf{y}^*$ and $\mathbf{u}^*$, then all these second order terms will be close to zero.

The $nT \times kT$ matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{u}}$ in (6)of the derivatives of the endogenous variables with respect to the controls is defined implicitly by the set of linear equations

$$\mathbf{J} \frac{\partial \mathbf{y}}{\partial \mathbf{u}} = -\frac{\partial \mathbf{f}}{\partial \mathbf{u}}. \tag{7}$$

The Gauss-Newton step in iteration s for updating the guess of $\mathbf{u}$ from the previous iteration, $\mathbf{u}^{s-1}$, is defined by

$$\left(\frac{\partial \mathbf{y}}{\partial \mathbf{u}}' \mathbf{W} \frac{\partial \mathbf{y}}{\partial \mathbf{u}} + \mathbf{Q} \right) (\mathbf{u}^s - \mathbf{u}^{s-1}) = -\frac{\partial \mathbf{y}}{\partial \mathbf{u}}' \mathbf{W} (\mathbf{y} - \mathbf{y}^*) - \mathbf{Q} \mathbf{u}^{s-1} \tag{8}$$

The advantage of the Gauss-Newton algorithm compared with other, more general, (e.g. quasi-Newton) methods for solving the minimisation problem (4) subject to (5) is that it only uses first order derivatives. Under certain circumstances, it can be shown to be quadratically convergent and, when the model (5) is linear, it converges in a single iteration. However, in other circumstances the Gauss-Newton step may not always be a descent direction. In these cases a modified version of the method is the Levenberg-Marquardt algorithm (Marquardt (1963)) which ensures that the step taken is always in a descent direction.

When the model to be solved is forward-looking and the solution is by a stacked Newton method, then it is straightforward to augment the L-B-J algorithm to define the derivative matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{u}}$ by solving the set of equations (7). All that is required is that the matrix of derivatives of the model equations with respect to the control variables $\frac{\partial \mathbf{f}}{\partial \mathbf{u}}$ is computed along with the standard first order derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$. When automatic derivative methods are being employed and the number of control variables is small relative to the number of endogenous model variables (as is usually the case), then the additional cost of this is trivial. Then, exactly the same recursions need to be applied to the matrix$\frac{\partial \mathbf{f}}{\partial \mathbf{u}}$ in (7) as are applied to the vector $f(y)$ in (3) in the L-B-J algorithm. Having solved for the derivatives $\frac{\partial \mathbf{y}}{\partial \mathbf{u}}$, the Gauss-Newton step can then be defined by solving the system of equations (8). This is a linear system of order $kT \times kT$ and so will usually be of much lower order than $nT \times nT$ system (7). However, sparse methods may also be applied here if necessary.

In efficient implementation of model solution methods, some automatic equation ordering is often used to reduce the size of the simultaneous equations block. Purely input variables that do not depend on the simultaneous variables (and so can be determined before the simultaneous equations are solved) can be eliminated. So can output variables that do not feed back into the simultaneous variables (and so can be determined after the simultaneous equations have been solved). It is important in the context of computing derivatives of targets with respect to controls to note that targets may well be output variables while control variables might well influence targets through input variables. Thus, the derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ do need to be computed for all model equations and not just those in the simultaneous equations block.

Finally, although the algorithm outlined here has been developed primarily in the context of optimal control where the model to be solved is forward looking, it can equally well be applied to the case of control with purely backward looking models. In this case, the marginal cost of the extra computations is greater since model solution with backward-looking models is usually done recursively. However, this disadvantage may still be outweighed by the speed of the Gauss-Newton method compared to other nonlinear control methods. In particular, for a linear model, the Gauss-Newton method will converge in a single iteration whereas other nonlinear optimisation methods may still take many iterations.

4 Conclusions

A new algorithm has been suggested for the solution of nonlinear quadratic optimal control problems using a (generalised) Gauss-Newton method with analytic model derivatives and applying efficient sparse matrix methods to solve efficiently for the derivatives of target variables with respect to controls. This algorithm has been implemented in version 3 of the software package *WinSolve*. Three alternative nonlinear methods were used for comparison: a version of the same algorithm using purely numerical first order derivatives, a quasi-Newton algorithm, (again using numerical first order derivatives) based on the Broydon-Fletcher-Goldfarb-Shanno variant of Davidon, Fletcher and Powell (Davidon (1959), Fletcher and Powell (1963), Broydon (1970), Fletcher (1970) and Shanno (1970)) and Powell's (1964) conjugate directions algorithm, which doesn't use derivatives.

In general, implementation of the algorithm proved to be a success. In comparison with the numerical derivative version of the same algorithm, the version using automatic derivatives was always much faster to converge, often by a factor of three or four times. When solving linear or nearly linear forward-looking models, both versions of the Gauss-Newton outperformed the other algorithms. However, when attempting to solve problems with very non-linear models, especially when starting from bad initial values, the Gauss-Newton algorithm was outperformed by the quasi-Newton method and on occasion failed to converge at all. This suggests that, even with the Marquardt modification, the algorithm is less robust than one that builds up information on the Hessian matrix. On the other hand, the Powell algorithm was always the slowest to converge, suggesting that some derivative information is important in optimisation problems involving a large number of variables.

References

[1] Armstrong, J., Black, R., Laxton, D. and Rose, D. (1998), 'A robust method for simulating forward-looking models', *Journal of Economic Dynamics and Control*, 22, 489–501.

[2] Boucekkine, R. (1995), 'An alternative methodology for solving nonlinear forward-looking models', *Journal of Economic Dynamics and Control*, 19, 711–734.

[3] Broydon, C.G. (1970), 'The convergence of a class of double-rank minimisation algorithms: 1. General considerations', *Journal of the Institute of Mathematics and its Applications*, 6, 76–90.

[4] Davidon, W.C. (1959), 'Variable metric method for minimisation', Research and Development Report ANL-5990, US Atomic Energy Commission, Argonne National Laboratories.

[5] Duff, I.S. (1977), 'MA28 – a set of Fortran subroutines for sparse unsymmetric linear equations', Report AERE R8730, HMSO, London:UK.

[6] Duff, I.S., Erisman, A.M. and Reid, J.K. (1986), *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford: UK..

[7] Fletcher, R. (1970), 'A new approach to variable metric algorithms', *The Computer Journal*, 13, 317–322.

[8] Fletcher, R. and Powell, M.J.D. (1963), 'A rapidly convergent descent method for minimisation', The Computer Journal, 6, 163–168.

[9] Griewank, A. (2000), *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA.

[10] Hollinger, P. (1996), 'The stacked-time simulator in TROLL: a robust algorithm for solving forward-looking models', mimeo, Intex Solutions, Needham, MA, USA.

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

[12] Juillard, M. (1996), 'DYNARE: a program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm', CEPREMAP working paper No. 9602, Paris, France.

[13] Juillard, M., Laxton, D., McAdam, P. and Pioro, H. (1998), 'An algorithm competition: first-order iterations versus Newton-based techniques', *Journal of Economic Dynamics and Control*, 22, 1291–1318.

[14] Laffargue, J-P. (1990), 'Résolution d'un modèle macroéconomique avec anticipations rationnelles', *Annales d'Economie et de Statistique*, 17, 97–119.

[15] Marquardt, D.W. (1963), 'An algorithm for least-squares estimation of nonlinear parameters', *Journal of the Society for Industrial and Applied Mathematics*, 11, 431–441.

[16] Newton, I. (1686), *Philosophiae Naturalis Principia Mathematica*, translation by A. Motte (1729), revised by F. Cajori (1934), University of California Press, Berkeley, CA, USA.

[17] Pierse, R.G. (2002), 'WinSolve: a users' guide', available at http://www.econ.surrey.ac.uk/winsolve/.

[18] Powell, M.J.D. (1964), 'An efficient method for finding the minimum of a function of several variables without calculating derivatives', *The Computer Journal*, 7, 155–162.

[19] Rall, L.B. (1981), *Automatic Differentiation: Techniques and Applications*, Springer Verlag, Berlin, Germany.

[20] Shanno, D.F. (1970), 'Conditioning of quasi-Newton methods for function minimisation', *Mathematics for Computation*, 24, 647–656.