No: 9602

DYNARE: A program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm.

Michel Juillard[1]

[1] Email: juillard@cepremap.cnrs.fr

DYNARE: un programme pour la résolution et la simulation de modèles dynamiques avec variables avancées à l'aide d'un algorithme de relaxation

RESUME: Le papier présente une généralisation de l'algorithme de Laffargue pour la simulation de modèles à anticipations avec des avances ou des retards sur plusieurs périodes et son implémentation dans le logiciel DYNARE.

DYNARE: A program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm.

ABSTRACT: This paper presents a generalization of Laffargue's algorithm for simulating rational expectation models with several leads and lags and its implementation in a Gauss program, DYNARE.

Mots clés: anticipations rationnelles, modèles non–linéaires, simulations.

Keywords: rational expectations, non–linear models, simulations.

J.E.L. classification system: C63, C88, E17

Since expectations, and particularly rational expectations, have become an important issue in macroeconomics, models including advanced values of dependent variables have started to appear. These models raise specific computational problems which have been dealt with in various manners: shooting methods, Gauss–Seidel and Gauss–Raphson type of algorithms. The program presented in this paper uses the last one. It builds on previous work by Laffargue (1990) and Boucekkine (1995).

The Gauss–Raphson algorithm suggests to stack up the equations of the models for all the periods in the simulation and to solve in block the resulting system. For medium to large models it requires solving a very large linear problem. Laffargue (1990) has shown that for models with one lag and one lead the problem can be reduced to manageable proportions by exploiting the special structure of the Jacobian matrix. The present paper extends Laffargue's algorithm for model with an arbitrary number of leads and lags. It is well known that a model with several leads and lags can be transformed through the use of auxiliary variables into an equivalent model with one lead and one lag, but, as will be argued in the following sections, the direct method is computationally faster than solving for the larger transformed model. In addition, we use in this paper a matrix–block presentation which let us then take advantage of a matrix–oriented programming language such as Gauss.

The algorithm presented in this paper is implemented in DYNARE, a program written in Gauss. In addition to the actual computation of the simulations, DYNARE provides the user with parsing tools so that the model can be written in an understandable manner with meaningful names for the variables and simple expression for leads and lags. The program creates automatically relevant matrices for the solving of the system. It also includes reporting and ploting routines.

In order to test the performance of the program, we have used as an example the German part of MULTIMOD, a interregional model developed at the I.M.F. (see Masson, Symansky and Meredith, 1990, and I.M.F., 1991). The model used here has 46 dependent variables, with lags ranging from 1 to 3 periods and leads from 1 to 5 periods. The simulation consisted in evaluating the consequences of an increase by about 30% of government expenditures in period 5. The simulation carries over 20 periods.

The first section of the paper details the problem to be solved and establishes notation. In the second section, we develop the solving algorithm for one lead and one lag. Then, we present the extension to several leads and lags. In the third section, we describe additional details to take advantage of known zero columns in the Jacobian.

1 Non–linear models with forward–looking variables

We consider non–linear models with n dependent variables, $y_{i,t}$, $i = 1, \ldots, n$, m independent or exogenous variables, $x_{j,t}$, $j = 1, \ldots, m$, and p parameters, θ_k, $k = 1, \ldots, p$. We use the notation y_t for the vector of dependent variables at time t, x_t, the vector of exogenous variables, and θ, the vector of parameters.

The model is such that the n current values of y_t depends on previous and future values of the dependent variables, the exogenous variables and the parameters. Exogenous variables may appear themselves with leads or lags, but this will not change the nature of the problem to be solved. For simplicity, we can consider only current exogenous variables without affecting the generality of the discussion.

For the current values of the dependent variables to be determined, there must be n equations, that we will express in homogenous form:

$$\begin{aligned} g_1(y_{t-r}, \ldots, y_t, \ldots, y_{t+s}, x_t, \theta) &= 0 \\ \vdots \quad & \quad \vdots \\ g_n(y_{t-r}, \ldots, y_t, \ldots, y_{t+s}, x_t, \theta) &= 0 \end{aligned}$$

In the above system, we consider a maximum of r lags and s leads. In most problems, not all of the n dependent variables appear at each of the $s + r + 1$ leads or lags. A basic identification condition requires however that they all appear as current variable. The absence of variables as some lead or lag can be exploited by the algorithm. It is a point on which we will return in section 3.

To a given sequence of exogenous variables, x_t, $t = 1, \ldots, T$, and given θ, will correspond a trajectory for the time span 1 to T. A particular trajectory will correspond to particular initial conditions $y^*_{1-r}, \ldots, y^*_0$ and particular terminal conditions $y^*_{T+1}, \ldots, y^*_{T+s}$[2]. Conditions for the existence of a single trajectory are discussed in Boucekkine (1995).

The system to be solved is as follows:

$$\begin{aligned} y_{1-r} &= y^*_{1-r} \\ \vdots \quad & \quad \vdots \\ y_0 &= y^*_0 \\ g_1(y_{1-r}, \ldots, y_1, \ldots, y_{1+s}, x_1, \theta) &= 0 \end{aligned}$$

[2] One can also consider other terminal conditions. See for example Boucekkine, Juillard and Malgrange (1994).

$$
\begin{array}{rcl}
\vdots & & \vdots \\
g_n(y_{1-r},\ldots,y_1,\ldots,y_{1+s},x_1,\theta) & = & 0 \\
\vdots & = & \vdots \\
g_1(y_{T-r},\ldots,y_T,\ldots,y_{T+s},x_T,\theta) & = & 0 \\
\vdots & & \vdots \\
g_n(y_{T-r},\ldots,y_T,\ldots,y_{T+s},x_T,\theta) & = & 0 \\
y_{T+1} & = & y^*_{T+1} \\
\vdots & & \vdots \\
y_{T+s} & = & y^*_{T+s}
\end{array}
$$

As, for a particular simulation, the exogenous variables x_t, $t = 1,\ldots,T$ and the parameters θ are given, one can rewrite the equations of the model making appear only the dependent variables and using a more compact writing:

$$
f_t(z_t) \quad = \quad \begin{bmatrix} g_1(y_{t-r},\ldots,y_t,\ldots,y_{t+s},x_t,\theta) \\ \vdots \\ g_n(y_{t-r},\ldots,y_t,\ldots,y_{t+s},x_t,\theta) \end{bmatrix} \qquad t = 1,\ldots,T
$$

where $z_t = \begin{bmatrix} y'_{t-r} & \ldots & y'_t & \ldots & y'_{t+s} \end{bmatrix}'$. For the initial and terminal conditions, we write:

$$
\begin{array}{rcl}
f_{1-r}(y_{1-r}) & = & y_{1-r} - y^*_{1-r} = 0 \\
\vdots & & \vdots \\
f_0(y_0) & = & y_0 - y^*_0 = 0 \\
\\
f_{T+1}(y_{T+1}) & = & y_{T+1} - y^*_{T+1} = 0 \\
\vdots & & \vdots \\
f_{T+s}(y_{T+s}) & = & y_{T+s} - y^*_{T+s} = 0
\end{array}
$$

Finally, we can build Y, the vector of the values of the dependent variables for all the periods piled–up.

$$Y = \begin{bmatrix} y_{1-r} \\ \vdots \\ y_0 \\ y_1 \\ \vdots \\ y_T \\ y_{T+1} \\ \vdots \\ y_{T+s} \end{bmatrix}$$

The entire system with $(T + r + s) \times n$ equations can be written as

$$F(Y) = \begin{bmatrix} f_{1-r}(y_{1-r}) \\ \vdots \\ f_0(y_0) \\ f_1(z_1) \\ \vdots \\ f_T(z_T) \\ f_{T+1}(y_{T+1}) \\ \vdots \\ f_{T+s}(y_{T+s}) \end{bmatrix} = 0$$

2 Solving the system with the Newton–Raphson method

At first glance, the method is quite simple: the system $F(Y) = 0$ is approximated iteratively. At each step, the vector Y is modified by an amount $\Delta Y = -\left[\frac{\partial F}{\partial Y}\right]^{-1} F(Y)$.

The difficulty with this approach resides with the size of the Jacobian $\frac{\partial F(Y)}{\partial Y}$, which is a matrix of dimension $[n \times (T + r + s)] \times [n \times (T + r + s)]$. In our example, for a 46 equation model simulated over 20 periods, it is a 1288 by 1288 matrix. However, as is shown in Laffargue (1990), the structure of this matrix is such that its triangularization can be handled recursively and that there is no need to store the entire Jacobian matrix at any time. In fact, as we will show below, only a matrix 920 by 14 needs to be stored in our example.

The triangularization could be programmed element by element, but experience as shown us that, when we use a matrix oriented language such as Gauss, there are gains both in program simplicity and in execution time if we handle the problem by block corresponding to one period. In the next section, we present the algorithm when there are only one lead and one lag. Then, we show the extensions necessary to handle an arbitrary number of leads and lags.

2.1 The algorithm for one lead and one lag

In this case, the system is:

$$F(Y) \;=\; \begin{bmatrix} f_0(y_0) \\ f_1(z_1) \\ \vdots \\ f_T(z_T) \\ f_{T+1}(y_{T+1}) \end{bmatrix} = 0$$

where z_t simplifies to $\begin{bmatrix} y_{t-1} & y_t & y_{t+1} \end{bmatrix}'$.

The equation for the computation of the improvement of Newton–Raphson can be written in a way which puts in evidence the particular structure of the Jacobian:

$$\begin{bmatrix} I & & & & & & & \\ S_{1,-1} & S_{1,0} & S_{1,1} & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & S_{t,-1} & S_{t,0} & S_{t,1} & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & S_{T,-1} & S_{T,0} & S_{T,1} & \\ & & & & & I & & \end{bmatrix} \mathbf{\Delta Y} \;=\; -\begin{bmatrix} 0 \\ f_1(z_1) \\ \vdots \\ f_t(z_t) \\ \vdots \\ f_T(z_T) \\ 0 \end{bmatrix}$$

where $S_{t,k}$, $k = -1, 0, 1$, are the partial Jacobians:

$$S_{t,-1} = \frac{\partial f_t(z_t)}{\partial y_{t-1}}$$
$$S_{t,0} = \frac{\partial f_t(z_t)}{\partial y_t}$$
$$S_{t,1} = \frac{\partial f_t(z_t)}{\partial y_{t+1}}$$

As already mentioned, the partial Jacobians $S_{t,-1}$ and $S_{t,1}$ have many empty columns and we can take advantage of this characteristics. However, we will leave aside this aspect for now and return to it in section 4.

The triangularization aims at eliminating the elements below the main diagonal and can proceed recursively from the top[3]. The handling of the first period is particular because of the initial conditions. We have the equation:

$$S_{1,-1}\Delta y_0 + S_{1,0}\Delta y_1 + S_{1,1}\Delta y_2 \;=\; -f_1(z_1)$$

The initial conditions give us $\Delta y_0 = 0$ and we can solve the linear problem[4] for Δy_1:

$$\Delta y_1 + S_{1,0}^{-1} S_{1,1}\Delta y_2 \;=\; -S_{1,0}^{-1} f_1(z_1)$$

[3]Invertly, we could as well eliminate the elements above the main diagonal and start at the bottom of the system.

[4]In practice, languages such as Gauss provide a primitive which computes the solution of a linear problem without computing the inverse.

After this first step, the system looks like:

$$\begin{bmatrix} I & & & & & \\ & I & C_1 & & & \\ & S_{2,-1} & S_{1,0} & S_{1,1} & & \\ & & \ddots & \ddots & \ddots & \\ & & & S_{T,-1} & S_{T,0} & S_{T,1} \\ & & & & & I \end{bmatrix} \mathbf{\Delta Y} = \begin{bmatrix} 0 \\ d_1 \\ -f_2(z_2) \\ \vdots \\ -f_T(z_T) \\ 0 \end{bmatrix}$$

For the second period, and all the following ones, the basic equation is:

$$S_{t,-1}\Delta y_{t-1} + S_{t,0}\Delta y_t + S_{t,1}\Delta y_{t+1} = -f_t(z_t)$$

We retrieve the value of Δy_{t-1} from the equation for the previous period:

$$\Delta y_{t-1} + C_{t-1}\Delta y_t = d_{t-1}$$

By substitution and eliminating the term $S_{t,-1}\Delta y_{t-1}$, we obtain

$$(S_{t,0} - S_{t,-1}C_{t-1})\Delta y_t + S_{t,1}\Delta y_{t+1} = -f_t(z_t) - S_{t,-1}d_{t-1}$$

Then, solving for Δy_t,

$$\Delta y_1 + (S_{t,0} - S_{t,-1}C_{t-1})^{-1}S_{t,1}\Delta y_{t+1} = -(S_{t,0} - S_{t,-1}C_{t-1})^{-1}(f_t(z_t) + S_{t,-1}d_{t-1})$$

After triangularization, the system looks like:

$$\begin{bmatrix} I & & & & \\ & I & C_1 & & \\ & & \ddots & \ddots & \\ & & & I & C_T \\ & & & & I \end{bmatrix} \mathbf{\Delta Y} = \begin{bmatrix} 0 \\ d_1 \\ \vdots \\ d_T \\ 0 \end{bmatrix}$$

where $C_t = (S_{t,0} - S_{t,-1}C_{t-1})^{-1}S_{t,1}$ and $d_t = -(S_{t,0} - S_{t,-1}C_{t-1})^{-1}(f_t(z_t) + S_{t,-1}d_{t-1})$. The value of ΔY are then easily obtained through backward substitution:

$$\Delta y_t = d_t - C_t\Delta y_{t+1}$$

Using this approach, the only blocks requiring storage are C_t and d_t, $t = 1, \ldots, T$. As already mentioned, further reduction in storage is obtained by taking into account the empty columns of the partial Jacobians.

2.2 The triangularization algorithm with several leads and lags

With r lags and s leads, the system used for the determination of ΔY has essentially the same structure, but with more blocks under and above the main diagonal:

$$\begin{bmatrix} I & & & & & & & & \\ & \ddots & & & & & & & \\ S_{1,-r} & \dots & S_{1,0} & \dots & S_{1,s} & & & & \\ & \ddots & & \ddots & & \ddots & & & \\ & & S_{t,-r} & \dots & S_{t,0} & \dots & S_{t,s} & & \\ & & & \ddots & & \ddots & & \ddots & \\ & & & & S_{T,-r} & \dots & S_{T,0} & \dots & S_{T,1} \\ & & & & & & & \ddots & \\ & & & & & & & & I \end{bmatrix} \mathbf{\Delta Y} \quad = \quad - \begin{bmatrix} 0 \\ \vdots \\ f_1(z_1) \\ \vdots \\ f_t(z_t) \\ \vdots \\ f_T(z_T) \\ \vdots \\ 0 \end{bmatrix}$$

The consequences of initialization carries over for the first r periods. For the first period, because $\Delta y_k = 0$, for $k = 1-r, \dots, 0$, we have simply

$$\begin{bmatrix} I & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & I & & & & & & & \\ & & & I & C_{1,1} & \dots & C_{1,s} & & & \\ S_{2,-r} & \dots & S_{2,-1} & S_{2,0} & S_{2,1} & \dots & S_{t,s} & & & \\ & \ddots & & & \ddots & & & \ddots & & \\ & & S_{T,-r} & \dots & S_{T,-1} & S_{T,0} & S_{T,1} & \dots & S_{T,s} & \\ & & & & & I & & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & & I \end{bmatrix} \mathbf{\Delta Y} \quad = \quad \begin{bmatrix} 0 \\ \vdots \\ 0 \\ d_1 \\ -f_2(z_2) \\ \vdots \\ -f_T(z_T) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $C_{1,k} = S_{1,0}^{-1} S_{1,k}$, $k = 1, \dots, s$ and $d_1 = -S_{1,0}^{-1} f_1(z_1)$.

For period $t = 2, \dots, r$, the previous values of Δy are partly set to zero by initial conditions, partly determined by previous computations. The recurrence is as follows:

$$\begin{aligned} Q_0 &= S_{t,1-t} \\ Q_i &= S_{t,i+1-t} - \sum_{j=\max(1,i-t+2)}^{\min(i,t-1)} Q_{j-1} C_{j,i+1-j} \quad i = 1, \dots, s+t-2 \\ C_{t,k} &= Q_{t-1}^{-1} Q_{t+k-1} \quad k = 1, \dots, s-1 \\ C_{t,s} &= Q_{t-1}^{-1} S_{t,s} \\ d_t &= -Q_{t-1}^{-1} (f_t(z_t) + \sum_{j=1}^{t-1} d_j \end{aligned}$$

From period $r+1$, the previous values of Δy are entirely determined by previous computations and the recurrence is:

$$Q_0 = S_{t,-r}$$

$$
\begin{aligned}
Q_i &= S_{t,i-r} - \sum_{j=\max(1,i-r+1)}^{\min(i,s)} Q_{t-r+j-1} C_{t-r+j,i+1-j} \quad i = 1,\dots,r+s \\
C_{t,k} &= Q_r^{-1} Q_{r+k} \quad k = 1,\dots,s-1 \\
C_{t,s} &= Q_r^{-1} S_{t,s} \\
d_t &= -Q_r^{-1}(f_t(z_t) + \sum_{j=1}^{r} Q_{r-j} d_{t-j}
\end{aligned}
$$

Once the Jacobian matrix is entirely triangularized, the backward substitution takes the form:

$$
\Delta y_t = d_t - \sum_{i=1}^{s} C_{t,i} \Delta y_{t+i}
$$

In the case with several leads or lags, we have to store the blocks $C_{t,k}$, $t = 1,\dots,T$, $k = 1,\dots,s$ and d_t.

3 Using the sparsity of the Jacobian blocks

Usually, not all variables of the model are present at each lead or lag. This implies that several columns of the Jacobian blocks are empty and those are known before starting the computation. One can take advantage of this feature in the following ways.

For one period, the argument of the function $f_t()$ needs only the subvector of z_t containing lagged and leading variables actually used in the model. Note that for the coherence of the model, all the variables must appear at the current period.

If the function $f_t()$ is redefined in this manner, its corresponding Jacobian will necessarily contain the non–empty columns of $S_{t,k}$, $k = -r,\dots,s$.

The following rules must be observed for the triangularization algorithm:

1. The variables present in block C_i, $i = 1,\dots,s-1$ are equal to the union of the variables of the block $S_{t,i}$ and of the blocks C_k, $k > i$. The consequence is that the C_i blocks will always be non–empty, even if no variables are present at certain leads.

2. The variables present in block Q_i are equal to the union of the block $S_{t,-r+i}$ and of the blocks C_k, $k = 1,\dots,min(i,r)$.

3. For periods 1 to $r-1$, the variables present in the Q_i are different from the general case and special care must be taken for the lags containing no variable at all.

Often the gain of taking into account empty columns can be quite substantial. In the example used for this paper, there are 46 variables. The maximum lead is 5 and the maximum lag, 3. In the absence of special treatment, the

Jacobian for one period would have $9 * 46 = 414$ columns and we would have to store C blocks with $5 * 46 = 230$ columns. Taking into account the empty columns reduces the size of the Jacobian to 86 columns and the matrix necessary for storing the C blocks has only 13 columns.

Furthermore there exists a possible choice between speed and storage. If the number of leading variables is smaller than the number of lagging ones, we will minimize storage by eliminating the lower submatrix in the triangularization. On the contrary, we will minimize computation and improve speed by eliminating the upper submatrix. The reverse applies if the number of leading variables is greater than the number of lagging ones. The effective importance of this feature of the algorithm has not yet been tested.

4 Adding auxiliary variables or using the direct algorithm?

As we mentioned earlier, any model with several leads and lags can be transformed into a one–lead one–lag model through the use of auxiliary variables. We can now consider which of this transformation or of the direct algorithm presented above is the most efficient to handle general models.

To eliminate a variable at lag or lead $k > 1$, one needs $k - 1$ auxiliary variables. Therefore, the number of columns necessary for the storage of the C blocks will be identical in both approaches. However, using auxiliary variables will also add to the number of lines of the matrices used in each period. That doesn't happen with the direct algorithm. In term of storage, the direct algorithm is clearly superior.

On the speed side, the direct algorithm requires additional computations to eliminate several blocks under (or above the main diagonal of the Jacobian matrix). On the other hand, the transformation approach needs solving a larger linear problem. For practical implementation, the comparison is made more difficult by the fact that one uses a Gauss language primitive to solve the linear problems when the direct algorithm requires loops coded in Gauss language which is slower. Obviously, the larger the model, the more costly it will be to add additional variables.

For example, on the example used through out this paper, one iteration of the Newton–Raphson algorithm for a simulation of 20 periods takes 23.1 seconds on a Pentium at 75 MHz when we transform the model by adding auxiliary variables and 19.7 seconds when we use the direct approach detailed here. The improvement in speed is therefore of about 15%.

5 Conclusion

This paper detailed the algorithm used in DYNARE to simulate dynamic models with forward–looking variables. It is an application the Newton–Raphson algorithm which takes into consideration the special structure of the Jacobian matrix in such models. We also show that direct treatment of several leads or lags models improves storage and speed requirement over the transformation of the model through adding auxiliary variables.

It remains however that an important part of computation time is taken by the numerical evaluation of the Jacobians in each period. A most noticeable speed improvement would be brought by the use instead of symbolic differentiation. It will be one of the directions pursued in this research.

References

Boucekkine, R. (1995) "An alternative methodology for solving nonlinear forward-looking models," *Journal of Economic Dynamics and Control*, forthcoming.

Boucekkine, R., M. Juillard & P. Malgrange (1994) "Precision performances of terminal conditions for short time horizons forward-looking systems," paper presented at the IFAC Conference, June 1994, Amsterdam.

I.M.F. (1991) "Changes to Multimod since the July 1990 Occasional Paper No. 71. Current model: Multiap" Washington, DC: I.M.F.

Juillard, M. (1995) "DYNARE, a program for the resolution of non–linear models with forward–looking variables. Release 1.3" Paris: CEPREMAP.

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

Masson, P., S. Symansky & G. Meredith (1990) "Multimod Mark II: A revised and extended model" I.M.F. Occasional Paper No. 71, July.