DSGE Nash: a toolkit to solve Nash Games in Macro Models *

Massimo Ferrari† Maria Sole Pagliari‡

Abstract

This paper presents DSGE Nash a toolkit to solve for the pure strategies Nash equilibria of policy games in macro models. Despite the toolkit is primarily design to solve for the Nash equilibria of policy games in DSGE models, it allows for a wide variety of settings. These include solutions up to the third order, multiple players and strategies and the use of user-defined objective functions. Thought the same option, any macro model (not necessarily solved through Dynare) can be used to compute the players' payoff matrix and the Nash equilibria of the game. Moreover, users can set as objective for players any output of the model, for example (but not limited to): second-order means, the volatility of variables, impulse responses to specific shocks or any convolution of them. Finally, when only one player is selected, the problem is re-framed as a standard optimal policy problem and DSGE Nash computes the values of the policy parameters that maximize the objective function, i.e. the optimal policy.

Keywords: DSGE model, optimal policies, computational economics

JEL Codes:

*The authors would like to thank Gauthier Vermandel for discussion, as well as Michele Ca' Zorzi, Georgios Georgiadis, Ivan Jaccard, Adrian Penalver, seminar participants at the ECB and Banque de France and an anonymous referee for useful comments and suggestions. The views expressed in this paper are those of the authors and do not necessarily reflect those of either the Banque de France, the European Central Bank or the ESCB.

†European Central Bank, Sonnemannstrasse 20, 60314 Frankfurt am Main. E-mail: massimo.ferrari1@ecb.europa.eu

‡Banque de France, 39 Rue Croix des Petits Champs, 75001 Paris. E-mail: maria-sole.pagliari@banque-france.fr.

1 Introduction

A key question in structural modelling is the determination of the "optimal policy", in other terms how strong policy makers should respond to fluctuations in key macroeconomic aggregates in order to maximize their objective function.

This question has been widely discussed in the literature on monetary, see Woodford (2003), and fiscal, see Chari et al. (1994), policy where the main concern is to determined the optimal response of the central bank or the government to inflation, output or other observable variables in order to maximize households' welfare. The problem can be solved either "analitically", by optimizing the objective function of the planner under the constraints given by the other equations in the model. Alternatively, the problem can be solved "numberically" by drawing values for the policy parameters and selecting those that give the highest objective function. The main technical difficulty of these exercises relates to the calculation of households' welfare, which crucially depends both on the levels and the volatility of its arguments (in most cases consumption and labor). Linear first-order solutions of macro models do not account for the impact of volatility on welfare and hence cannot deliver accurate welfare metrics for policy analysis. Two solutions have been found. On the one hand Benigno and Woodford (2012) and Benigno and Woodford (2004) show how to construct a linear-quadratic (LQ) approximation of the policy function. Their method allows to construct second-order accurate welfare values based on first-order approximations. While the approach is elegant and the approximation straightforward to implement, it is often extremely computational intensive to calculate the LQ approximation in for very larger models. The alternative solution, used between others by Born and Pfeifer (2020), is to directly solve the model at second-order and compute the mean of the objective variable using its state-space representation as in Kim et al. (2008). This approach has the advantage to be extremely straightforward to implement once the solution of the model is automatized.

The quest for optimal policy parameters is more complicated once multiple players are involved. This is a typical problem in international macro models where, for example, multiple central banks or government have to choose *contemporaneously* the optimal values of their policy functions. Clearly, the the main issue is that agents, when optimizing their own policy rules, need to take into account the strategic responses of other players. One player might act strategically avoiding specific strategies in anticipation of the opponents' reaction or to try to shift the social costs of specific policies to her neighbours. The final result is the Nash equilibrium of a game played in pure strategies by all optimizing agents. Clarida et al. (2002) and Banerjee

et al. (2016) solve this problem analytically by explicitly solving the optimal control problem of each agent: each problem optimizes the parameter of his policy function, in order to maximize welfare, having all other equations in the model as constraints (Bodenstein et al. (2019) recently developed a toolkit to compute this solution). The resulting first order conditions, including the solution for all the Lagrangian multipliers, are then added to the model. This approach involves computing numerical derivatives for each player's problem and including in the model file one Lagrangian multiplier per constrain per player; clearly this process can become fairly complex even in medium-size models.

DSGE Nash, instead, solves the problem using numerical methods. In particular, the toolkit solves for the Nash equilibrium, in pure strategies, of policy game with N players and strategies. The solution algorithm we use is intuitive and very simple. The software first computes the payoff matrix of the game; than, given that matrix, it (efficiently) solves for the Nash equilibria of the game, i.e. those combinations of strategies that are contemporaneously optimal for all players. There are several advantages of using numerical methods. First, in large models with more than two players, they might be faster and more feasible than numerical differentiation; second, this methods allow to target *any* objective variable including, for example, a LQ approximation of the policy function (if available), second or third moments of endogenous variables, impulse responses or any convolution of them. A third advantage is that the algorithm can be applied to any model, even those for which a closed-form solution is not available. On a more computational ground, our algorithm is fully parallelizeable, taking full advantage of the capabilities of modern server machines.

DSGE Nash can be used in three main configurations:

- **Policy equilibria in DSGE models (default):** the model is solved with Dynare and the algorithm computes the payoff matrix using a second-order approximation of the model and then the Nash equilibria of the game. As an option, if users can provide a LQ approximation of the objective function, the model can be solved at first order and the LQ solution is used to impute values in the payoffs matrix.

- **Policy equilibria in macro models:** DSGE Nash allows users to use a custom-made function to compute payoffs for each draw of the parameters of the policy functions. This is extremely easy to do, as it is sufficient to execute the user-defined code in a dedicated .m file. This allows to apply our solution algorithm to all model where strategic interactions are relevant.

- **Standard optimal policy:** if only one player is selected, DSGE Nash automatically re-

casts the problem into a standard optimal policy problem. In this case the toolkit simply computes the value of parameters that maximize the objective function of the player. Also in this case, the baseline is a second-order solution of the model through Dynare, but also in this case user can easily adapt write custom-made objective functions.

Our toolkit is similar in purposes but differs from Bodenstein et al. (2019) in five ways. First, Bodenstein et al. (2019) solves for the Nash equilibrium by computing the numerical derivatives of the objective function of each player when all other equations in the model and the policy functions of other players act as constraints. Our toolkit, instead, uses simulated methods to fill the payoff matrix of each player and then finds the Nash equilibria. There are benefits and costs from each approach. Computing numerical derivatives has the advantage of finding a closed-form solution for the policy problem. However, it can be very resource intensive, especially in large models with many variables. Moreover, their approach requires to add all the lagrangian multipliers (one set for each player) to the set of endogenous variables, which, when multiple players are involve, could rapidly increase the number of equations included in the model making it hard to solve expecially at higher orders. Our approach, on the other end, does not require to add more equations to the model and simply relies on simulated methods to compute payoffs for each combination of strategies. This can also be computational intensive for large models, but there are ways to reduce the dimensionality problem and the entire algorithm is highly parallelizeable. Second, our toolkit is not constrained in accepting endogenous variables only as players' objectives. We allow agents to target anything that can be computed from the model, for example the volatility of specific variables or the response to a specific shock. This greatly extends the range of experiments that can be run with out toolkit. Related to this, through the use of the user-defined objective function, our toolkit can also be easily adapted to models with non-linearities such as occasionally binding constraint, as in Holden (2019) and Cuba-Borda et al. (2019), which are becoming increasingly popular in the literature. Fourth, we allow for non-DSGE models or models non solved through Dynare to be used to compute players payoff. This last feature greatly extends the potential application of our code. Fifth, when only one player is selected, the toolkit automatically re-writes the problem into a standard optimal policy problem, thus significantly reducing the barriers for researchers to approach this type of analysis. All in all, our toolkit differs considerably from Bodenstein et al. (2019) despite being motivated by the need to answer to similar questions. We believe that the two toolkits complement each other as they offer different solutions to the same problem. Depending on the practical difficulties faced, researcher can chose the one that best fits their application.

2 The Nash problem

3 Solution strategy

3.1 Intuition

Consider the most well-known example of a game: the prisoner's dilemma. In the prisoner's dilemma two criminals (P1 and P2) need to decide whether to confess (C) the crime they committed or to stay silent (S). If they both confess, they are sentenced to 5-years of prison; if they both stay silent, the persecutors do not have a full account of their deeds, hence they are sentenced to just 2 years each. If only one of them confesses, he is able to shift the blame to his associated. The confessing criminal is set free and the other is severely punished with 10-years of prison. The cooperative solution is reported Table 1.

Clearly, the cooperative solution maximizes the *joint* welfare of the criminals, because they jointly serve only 4 years of detention. The key insight of game theory is that because P1 and P2 are selfish and cannot collude the "optimal" solution is not reached. Criminals know that is not in the best interest of their associate to stay silent if they do not confess. On the contrary, each player has always the incentive to confess no matter what the other does. If P2 confess, it is optimal for P1 to confess as well because otherwise he would serve 10 years of prison. If P2 stays silent, then P1 finds optimal to confess in order to be set free.

This simple example gives the intuition of the main problem of strategic interactions: individual incentives shape agents' decisions and the combined outcome of such choices might not be optimal from a global prospective.

Table 1: The prisoner's dilemma

		P2	
		C	S
P1	C	(5, 5)	(0,10)
	S	(10, 0)	(2,2)

3.2 Solution algorithm

Consider the problem of a player i who wants to maximise the objective variable W, using the instrument vector τ_i, conditional on the response of other players $-i$ and the structural equations of the model. The problem of i can be written more formally as:

$$\max_{\tau_i} E(W_{i,t})|(\tau_{-i}, \Omega) \tag{3.1}$$

where Ω defines the set of structural equations of the model and shocks. The Nash equilibrium of the game is defined as the intersection of the optimal response function of each country, i.e. the sequence of strategies $\{\tau_i|\tau_{-i}\}$ that solves Equation (3.1) for each possible strategy of the opponent. Practically, these are computed with the following algorithm:

1. for each combinations of opponent's strategies τ_{-i} compute the optimal response of i, i.e. the value of τ_i that maximises the objective $E(W_{i,t})$. This is the optimal response of i to the strategies $-i$;

2. we repeat this for all combinations of strategies τ_{-i}; and collect them in the vector $\tau_i^*|\vec{-}i$ which describes the optimal policy response function for i for each τ_{-i} of the other players;

3. we construct the vector of strategies T^* which matches the optimal response of i $\tau_i^*|\vec{-}i$ with the strategies of other players $-i$ for which $\tau_i^*|\vec{-}i$ is the best response;

4. if there is a Nash equilibrium of the game, it must be in T^*;

5. we then test whether each combination of strategies in the set T^* is indeed optimal for all players. If it is, it is a Nash equilibrium.

This procedure computes efficiently the intersection between the best reply functions of each player leveraging on the fact that if a set of strategies is a Nash equilibrium of the game, it must appear in all best reply functions. Because of this, it is sufficient to focus on the best reply function of the first player and test whether there is a combination of strategies therein that is indeed optimal for all players, i.e. it is an intersection between the best response functions of each player.

4 Application to an open-economy model

5 Conclusion

References

[1] Banerjee, R., Devereux, M. B., and Lombardo, G. "Self-oriented monetary policy, global financial markets and excess volatility of international capital flows". *Journal of International Money and Finance*, 68(C):275–297, 2016.

[2] Benigno, P. and Woodford, M. "Optimal Monetary and Fiscal Policy: A Linear-Quadratic Approach". In *NBER Macroeconomics Annual 2003, Volume 18*, NBER Chapters, pages 271–364. National Bureau of Economic Research, Inc, May 2004.

[3] Benigno, P. and Woodford, M. "Linear-quadratic approximation of optimal policy problems". *Journal of Economic Theory*, 147(1):1–42, 2012.

[4] Bodenstein, M., Guerrieri, L., and LaBriola, J. "Macroeconomic policy games". *Journal of Monetary Economics*, 101(C):64–81, 2019.

[5] Born, B. and Pfeifer, J. "Uncertainty-driven Business Cycles: Assessing the Markup Channel". *Quantitative Economics*, forthcoming, 2020.

[6] Chari, V. V., Christiano, L. J., and Kehoe, P. J. "Optimal Fiscal Policy in a Business Cycle Model". *Journal of Political Economy*, 102(4):617–652, 1994.

[7] Clarida, R., Galì, J., and Gertler, M. "A simple framework for international monetary policy analysis". *Journal of Monetary Economics*, 49:879–904, 2002.

[8] Cuba-Borda, P., Guerrieri, L., Iacoviello, M., and Zhong, M. "Likelihood evaluation of models with occasionally binding constraints". *Journal of Applied Econometrics*, 34(7): 1073–1085, 2019.

[9] Holden, T. D. "Existence and uniqueness of solutions to dynamic models with occasionally binding constraints". Technical report, 2019.

[10] Kim, J., Kimb, S., Schaumburg, E., and Sims, C. A. "Calculating and using second-order accurate solutions of discrete time dynamic equilibrium models". *Journal of Economic Dynamics and Control*, 32:3397–3414, 2008.

[11] Woodford, M. *Interest and Prices: Foundations of a Theory of Monetary Policy*. Princeton University Press, Princeton, N.J., 2003.

Appendix

A User Manual

A.1 Presentation

This notes present DSGE Nash, a toolkit to compute the Nash equilibrium, in pure strategies, of strategic games in macro models. Strategies are entered as grids over which DSGE Nash solves the model. The toolking default options are based on Dynare and on the standard optimal policy problem in DSGE models: users should give the name of one objective variable per player and one set of policy parameters per player; the toolkit solves the model at second order and uses the second-order steady state of the objective variables as payoffs.

The toolkit, however, allows user to specify custom functions as objective this allows: i) to use a first order solution and a LQ approximation to compute the objective functions; ii) to allow players to target any objective, for example volatilities or impulse response functions; iii) to use *any* model as input, non-DSGE models or DSGE models solved without using dynare, as long as they allow to compute one objetive function value for player. Moreover, when only one player is selected, the toolkit solves the standard optimal policy problem.

If Dynare is used, it should be version 4.6 or earlier.
To install DSGE Nash simply add the folder $\texttt{dsge}_{n}ash_{f}iletotheMatlabpath.$

A.2 Preliminaries

There are two preliminary steps to be made before using DSGE Nash:

1. If a Dynare mod file is used, users should end it with the `stoch_simul()` command, where the option `order` should be included. This makes sure that the correct order of approximation is used in DSGE Nash.

2. If Dynare is not used at all or other calculations are needed to compute the objective functions, users should have those computations done in the `user_defined_function.m`. In this file any calculations can be made, provided that the file assigns a value to the objective function of players.

3. Needless to say, the policy parameters need to appear in the mod file or in the user defined objective calculations. Similarly, a value needs to be assigned to the objective functions.

Hint: when used in DSGE Nash, models could be "optimized" to increase computation speed. One way, is to substitute redundant equations. For example, when writing ther recursive welfare function $W_t = U_t + \beta E_t(W_{t+1})$, user should directly substitute in the period utility and remove U_t from the endogenous variables.

Hint: several Dynare options should also be used. Pruning should be selected at higher solution orders; output should not be printed on screen (`noprint`) and graphs not created ((`nograph`)). Generally speaking, unless used in the user defined function, autoregressive coefficients (`AR=0`), imppulse responses (`irf=0`) and simulations (`periods=0`) are not needed. When the second-order steady state is targeted, the final line of the mod file should look like this: `stoch_simul(order=2,pruning,IRF=0,AR=0,nograph,noprint);`.

DSGE Nash is executed by running `dsge_nash(useroptions)`. `useroptions` is a structure that contains all the settings needed to run the toolkit. If `useroptions` is empty, the toolkit will ask users to input the settings through the terminal

A.3 Settings for policy games based on second order stochastic steady state of DSGE models

This is the default setting of DSGE Nash where the second-order mean of endogenous variables is used as objective. In a nutshell, DSGE Nash solves the model for a given combination of the policy parameters and extracts the second-order mean of the target variables from `oo_.mean` structure generated by Dynare. An example of the default settings is available in the example file `EXAMPLE1.m`. The settings are the following:

- `modname`: a string with the mod file name
- `usedynare`: =1 if Dynare is used to solve the model, 0 otherwise
- `nplayers`: number of players
- `parallel`: =1 to parallelize computations, 0 otherwise. The number of workers in the parallel pool should be set in the matlab preferences
- `userdefined`: =0 if the objective are second-order means of Dynare, =1 if a user-defined function is used to compute the objective
- `ovveridedynare`: =1 to change the default dynare settings with user-defined settings; 0 otherwise

- user defined options are the following: `pruning` =0/1 to use pruning; `irf`=0 for no IRFs; `ar`=0 for no AR coefficients; `nograph`=1 for no graphs; `noprint`=1 for not outputing on screen Dynare calculations; `periods`=0 for no simulations

- `objname`: a cell array with the name of the objective variable for each player, for example `{'OBJ1';'OBJ2'}`

- `instname`: a cell array with the list of instruments for each players, for example `instname{1,1}={'INST1_1','INST2_1'}` for the first player and `instname{2,1}={'INST1_2'}` for the second player

- `grid`: grid of values for each parameter, for example `grid{1,1}(1,1)={[0:0.2:4]}` and `grid{1,1}(1,2)={[0:0.2:4]}` for the first player and `grid{2,1}(1,1)={[0:0.05:0.8]}` for the second player

DSGE Nash proceeds via a grid search to fill the payoff matrix, i.e. computing the payoff of each player given a specific combination of their strategies. Notice that when a player can choose more than one instrument, a strategy for that player is given by a value for each of his instruments.

A.4 Settings for policy games based other user defined objective functions

Instead of setting as objective the second-order mean of endogenous variables (the standard choice in optimal policy problems) users can decide to compute themselves the objective variable for some of the players. Users can, for example, target: the volatility of variables, specific impulse responses, compute the LQ approximation of endogenous variables if available.[1]
These user-defined target variables should be computed in `user_defined_function.m`; an example is given in `EXAMPLE2.m` where user the objective is a weighted average of inflation and output volatility. Setting are identical to the previous case with the exception of:

- `userdefined`: =1

- `simlist`: a cell array of variables to be Dynare needs to simulate for the user-defined calculations. By default DSGE Nash simulates only the players' target variables. If users choose a user-defined simulation, and need to simulate a set of variables from the model, they should list them in this field. For example `simlist = { 'NY';'NPI';'NPE'}` ; to simulate all variables in the model use `simlist = { }`

- if IRFs need to be computed, then `irf`¿0; if data simulations are needed `periods`¿0

[1] This has the advantage of solving the model at first order only, making computations significantly faster.

- if user do not want to use Dynare at all, they should put all the compuations needed to get the value of the objective variables in the file `user_defined_function.m` and select `usedynare`=0

Hint: if second order solutions are not needed, users should end the mod file by setting `stoch_simul(order=1)`.

A.5 Settings for simple optimal policy exercise

If only one player is selected, DSGE Nash re-casts the problem as an optimal policy problem. The code computes the value of parameters that maximise the value of the objective variables. This is the standard welfare-optimization exercise. An example of this option is given in `EXAMPLE3.m`. Additionally to baseline options users need to provide:

- `usemin`: =1 if users want to use a minimization algorithm to solve the policy problem (in this case the min and max values of the parameters' grid are use as bounds for the optimizer); =0 if users want to use a grid search
- `minoptions`: options for the optimizer; users can set any options available in `Matlab`. Default is `optimset('Algorithm','interior-point','AlwaysHonorConstraints','bounds', 'TolFun', 1e-5,'TolX', 1e-5,'MaxFunEvals',100000000,'MaxIter',100000000, 'PlotFcns',@optimplotfval)`

Hint: also in this case users can compute themselves the target variable by using user-defined computations either based on the Dynare output (`userdefined=1`, `usedynare=1`) or not (`userdefined=1`, `usedynare=0`).

A.6 Output

In the case of a policy game, all Nash equilibria are printed on screen, i.e. the set of equilibrium strategies and players' payoffs. In case a simple optimal policy exercise is run, the optimal parameters value is printed instead of the strategies.
Results are also stored in the current folder. `OutputGridSearch.mat` stores the grid search output (i.e. the payoff matrix); `OptimalResponseFunctions.mat` saves the optimal response functions of each player (i.e. the best response of each player given a specific combination of other players' strategies, which are called O_RES1, O_RES2...); `N_OUT.mat` a structure with the DSGE Nash output.

A.7 Options

Here are reported all options that can be specified in the structure `dsge_nash(useroptions)`.

- `ar`: length of AE cofficients, =0 (default)
- `grid`: cell array with the set of possible values of each instrument. In case the optimizer is uses the max and min of these grids are used as bounds of the optimization
- `instname`: cell array with names of instruments for each player
- `irf`: irf length, =0 (default)
- `keepout`: =1 to keep all simulation results in the final structure, =0 if not
- `minoptions`: optmizer options, default `optimset('Algorithm','interior-point', 'AlwaysHonorConstraints','bounds','TolFun', 1e-5,'TolX', 1e-5,'MaxFunEvals', 100000000,'MaxIter',100000000,'PlotFcns',@optimplotfval)`
- `modname`: string with dynare mod file name
- `nograph`: no graph option, =1 (default)
- `noprint`: no graph print, =1 (default)
- `nplayers`: number of players
- `objname`: cell array containing the names of the objective variables of each player
- `ovveridedynare`: =1 to override Dynare options, =0 (default) if not
- `parallel`: =1 to parallelize computations, =0 (default) if not
- `pruning`: =1 (default) for pruning, =0 if not
- `seed`: random number seed, default 1999
- `simlist`: a cell array of variables to be Dynare needs to simulate for the user-defined calculations
- `usedynare`: =1 (default) to use Dynare to solve the model, =0 if not. In this case the model solution and the assignment of objective variables needs to be done in `user_defined_function.m`
- `usemin`: =1 (default) to use a minimization algorithm when single-player optimal policy, 0 for grid search

- `userdefined`: =1 if the objective variables are computed manually based on the Dynare ouput in the file `user_defined_function.m`, =0 (default) if not