PM A procedure for taking a model from Dynare to YADA quickly

DATUM: 2017-02-27
AVDELNING:
HANDLÄGGARE: Ingvar Strid
HANTERINGSKLASS Ö P P E N

SVERIGES RIKSBANK
SE-103 37 Stockholm
(Brunkebergstorg 11)

Tel +46 8 787 00 00
Fax +46 8 21 05 31
registratorn@riksbank.se
www.riksbank.se

DNR [Diarienummer]

Taking a model from Dynare to YADA

Purpose

The purpose of this memo is to describe one way to implement a model with Dynare and then transfer the model to YADA. Dynare is a good program for model development, experimentation and estimation, while YADA has its main strengths in the wide range of analysis tools, e.g. forecasting, decompositions etc.

Background

In 2013 we constructed a version of Ramses with a time-varying neutral rate. The model was implemented in Dynare and then transferred to YADA. Some 'tricks' were involved and the procedure was simplified by the fact that the extended model was fairly 'close' to the baseline model/code in terms of its representation. We have some notes on the procedure in the SW document *DynareInYADA_130830*. Since then the Ramses baseline model/code has changed, as described in the memo *RamsesCodeUpdate_20141029*. The main thing about the update is that the i) calibration and iI) estimation versions of the Dynare code were merged. One implication is that the 'discrepancy' (ordering and number of variables in the model) increased.

Now we want do describe and document the procedure more carefully with the goal that the procedure should be possible to use for any model.

The model in Dynare

It is assumed that the model has been implemented in a mod file in Dynare. For the procedure to work *it is crucial that the model is implemented in Dynare with maximum one lead and maximum one lag (1-1 form)*. This is 'good modeling practice' in any case, and it is a necessary with YADA. We can always setup the model in 1-1 form by introducing additional variables. Another drawback of not using the 1-1 form is that Dynare will introduce 'auxiliary state variables' (with obscure names…) which makes it harder to understand what is going on. It is preferable, we believe, to know the variables of the model.

Example: Ramses with a time-varying neutral rate

The model has been implemented in Dynare, the main model file is **R2_170227_tvr.mod** with accompanying steady state file **R2_170227_tvr_steadystate**. The model is described in a separate memo.

- Number of variables: M_.endo_nbr=215 (212)
- Number of innovations: M_.exo_nbr=43 (42)

where the numbers in parenthesis are the number of variables in the Dynare baseline version of Ramses. The TVR version contains the three additional variables data_RtvU, lRtvU and lRtvzU, and the additional innovation `Rtvz_eps`.

We compare the Ramses Dynare baseline model to the version of Ramses currently implemented in YADA which contains 224 variables.[1]

Variables that feature in the baseline Dynare model but not in YADA (4):

- data_unempU, data_qU, lOm2_1U, lOm2_2U

Variables that feature in YADA but not in the baseline Dynare model (16):

- lystarlag1U, lpistarlag1U, lRstarlag1U, lystarlag2U, lpistarlag2U, lRstarlag2U lystarlag3U, lpistarlag3U, lRstarlag3U, lystarlag4U lpistarlag4U, lRstarlag4U lzetatildeU, lRRstarU, lRRU, lLPU

Now, 212=224+4-16.

State-space representation in Dynare

In a separate memo, 'State space representation of a model in Dynare', we describe how to obtain the state-space representation, (F,B0), in Dynare.

The model in YADA

The model is calibrated/parameterised and solved using Dynare (see above), and then the solution (F,B0) is directly read/imported in YADA. Now we consider all the files we need to adapt when taking the model to YADA.

Solution of the model

Solving the model: KleinSolver.m

Our implementation is to import the solution from Dynare directly to YADA. The function …\gensys\KleinSolver.m in YADA produces exactly the matrices (F,B0). Our procedure in YADA then is:

- Solve the model in Dynare and obtain the state-space representation (F,B0), that is the numerical solution for a given parameterisation of the model. Instead of

[1] To be clear, the YADA model code used e.g. in MPR February 2017.

importing the numerical solution one could also solve the model using Dynare within YADA. However, we prefer not to run Dynare within YADA.

- In YADA choose to use the Klein solver as the solution method (in menu, or when setting up the .dsge file, see below).
- Modify KleinSolver.m. Instead of actually solving the model as in the original code, simply read the matrices (F,B0) which have been saved from Dynare. This essentially means that the function is replaced by a fake version which just contains a load command.

Calibration and prior: excel file

The calibration of the model and the prior distribution is provided in **an excel file with a name and location chosen by the user**, which is also referenced in the YADA menus. Importantly, note that the parameters in this file will not be effective, since parameterisation is done in Dynare. Hence, no action is required.

Additional parameters: *ModelName*SteadyStateParameters.m

In YADA it is possible to define a file with additional, derived, parameters, e.g. for baseline Ramses it contains steady state computations and some of the parameters are used in constructing the observation equation. With our approach the file may be empty.

Three additional files: compute_aim_data, compute_aim_matrices, and *ModelName*.aim

YADA has two functions which generate quantities that are needed for solving the model. Note that we are not going to use YADA to solve the model but we still need to consider these. Consider the following two functions in YADA:

- function [param_,np,modname,neq,nlag,nlead,eqname_,eqtype_,endog_,delay_,vtype_] = **compute_aim_data**()
- function [g,h] = **compute_aim_matrices**(ModelParameters)

Our version of **compute_aim_data** uses output from Dynare (**dynareOutputXXX.mat**) as input to generate the necessary outputs from this function. The Dynare objects needed are M_, oo_, F and B0. The output file is generated when we solve the model with Dynare. We are gonna use this version of **compute_aim_data** to generate a .mat file that YADA needs. The .mat file should contain the outputs:

- `param_,np,modname,neq,nlag,nlead,eqname_,eqtype_,endog_,delay_,vtype_`

Our version of **compute_aim_matrices** simply defines the matrices g and h as zero matrices. *This is where we differ from how the model has been transferred to YADA previously (Dynare to YADA, Laséen)*. Previously this function contained analytical expressions for the matrices g and h (with baseline Ramses some 1000 lines of code). Now we do not need these since we are not going to solve the model using YADA.

Finally we need a file where the model is normally written down ***ModelName*.aim**, which is empty in our case since we do not write the model in YADA.

Example: Ramses with a time-varying neutral rate

This is what we do for the example model:

- Solution: insert a load command of the file containing F and B0 in KleinSolver.m
- Calibration and prior: these excel files are not needed.
- Additional parameters: construct an empty m file.
- Compute_aim_matrices: empty file
- Compute_aim_data: insert a load command (same as for solution above) and generate the mat file *ModelName*.aim

DSGE file: *ModelName*.dsge

The .dsge, or model, file contains a number of settings. When a user chooses various settings in the YADA menus and saves the model file, this file will be dynamically updated. But it is also possible to edit the file directly (e.g. using notes). Using our approach we believe it is easier to edit the file using a text editor.

We need to edit the following rows in the .dsge file:

- Input data
    - DataConstructionFile: dir for file
    - MeasurementEquationFile: dir for file
    - PriorFile: dir for file
    - PriorFileSheet: name of sheet
    - UpdateParameterFile: dir for file
    - AIMFile: dir for file
    - NameOfModel: name of model
    - ModelSolver=x : x = 2 means Klein solver is used
    - OutputDirectory: name of dir
- Variable data
    - StateShockNames: list names of innovations, separate by $
        - Names as in M_.exo_names
    - StateShockPositions: list, #states+1 to #states+# innovations
        - # innovations= M_.exo_nbr
    - StateVariableNames: list names of variables, separate by $
        - Names as in M_.endo_names
    - StateVariablePositions: list, 1 to the number of variables (=#states)
        - # variables=#states= M_.endo_nbr
    - StateEquationPositions: list, 1 to the number of equations
    - ShockGroupNames: same as StateShockNames (or leave empty)
    - ShockGroups: list, 1 to # innovations (or leave empty)

Example: Ramses with a time-varying neutral rate

We insert the file paths in ‘input data’. Initially we use the same DataConstructionFile, MeasurementEquationFile, PriorFile and PriorFileSheet as for baseline Ramses since we have the same set of observable variables. If we want to use the neutral rate as an

observable in YADA it is easy to modify the DataConstructionFile and MeasurementEquationFile.

Then we use outputs from Dynare, i.e. lists of variables and innovations, to edit 'variable data'. The .dsge file contains many more fields but we do not need to edit these since they will be dynamically changed once we make selections in the YADA menus once the model is up and running.

YADA input file: DataConstFile.m

The DataConstFile can be written e.g. by benchmarking on the one for baseline Ramses.

Observation equation: MeasurementEqFile.m

The measurement equation can be written e.g. by benchmarking on the one for baseline Ramses.

YADA input and output

The changes needed to the I/O will depend on the model implemented, at it could potentially involve writing a lot of new code.

Example: Ramses with a time-varying neutral rate

For this model we use the same data on observed variables as with baseline Ramses, which means that the additional works is minimal.

- PostHandlingData: an *excel file with the names of the state variables* is read, needs to be updated
- ForecastToolsMenuFunctions: did some changes related to how innovations are represented in YADA, an 'old issue' that sometimes creates problems. Related e.g. to CompareMulti.
- CompareMulti: did some changes to be able to plot innovations correctly.