# USER GUIDE FOR GUERRIERI-IACOVIELLO OCCBIN TOOLKIT

JANUARY 29, 2014

CITE AS: Guerrieri, Luca, and Matteo Iacoviello (2014) "Occbin: A Toolkit to Solve Models with Occasionally Binding Constraints Easily", working paper, Federal Reserve Board

## Overview

Modify and run the file `setpathdynare4.m` so as to point to the local Dynare installation directory and to the directory containing the `toolkit_files`.

We have used successfully Dynare versions 4.3.1 on Windows and 4.3.3 on Mac.

## List of Example Files

The first three examples refer to the three models described in the paper.

1. `runsim_irrcap.m.` An RBC model with a constraint on the level of investment. The two relevant mod files are dynrbc.mod and dynrbcirr_i.mod.

   This examples shows how to declare the parameter values in an extermal file (named paramfile_irrcap.m, called from dynrbc_steadystate.m).

   If the example file runs correctly, it will generate a figure like the one in the file figure_example.pdf

2. `runsim_newkeynesian.m` In this example, `fv.mod` is the model with the ZLB described in the paper. Parameter values are declared in the file paramfile_fv. These parameter values can be overwrittem in the runsim_newkeynesian file.

3. `runsim_borrcon.m` In this example, `borrcon.mod` is a model with borrowing constraint that binds occasionally. Parameter values are declared in the file paramfile_borrcon. These parameter values can be overwrittem in the runsim_borrcon file.

   Additional examples.

4. `runsim_irrcap_twoconstraints.m.` An RBC model with a constraint on the level of investment and an asymmetric capital adjustment cost. This file shows how one can use the codes to solve a model with two occasionally binding constraints using the function `solve_two_constraints`

5. `runsim_irrcap_twoconstraints_computepolicy.m.` For the same model as above, but uses the code and the initcon option to show one can use the toolkit to derive the model's nonlinear policy functions.

6. `runsim_cgg.m`. Solves a version of the Clarida-Gali-Gertler ("The Science of Monetary Policy: A New Keynesian Perspective, JEL, 2009) model allowing for an inertial Taylor rule subject the zero lower bound on nominal interest rates (see equation for `rnot`, the notional interest rate).

7. `runsim_smetswouters.m`. Solves the Smets-Wouters (AER, 2007) model allowing for the zero lower bound on nominal interest rates (see equation for `rnot`, the notional interest rate). The codes for the model were downloaded from the online Appendix on the AEA webpage. We affected minimal changes to the code for compatibility with Dynare 4. To avoid the estimation step, some parameter values were set at their initial level prior to estimation.

8. `runsim_dnk.m.` Solve a new-keynesian model with zero lower bound and government spending. This folder shows how one can use the codes to declare the parameter values only once in an outside file (named paramfile_dnk.m). Shows how one can use separate sets of functions to solve model disregarding nonlinearities, or to compute impulse responses conditional on different baseline paths for the variables.

   – `dnk.mod` contains a standard new-keynesian model specified away from the zlb constraint.

   – `dnk_zlb.mod` is an exact replica of dnk.mod file with the model specified at the constraint.

   Except for the interest rate equation, the models in the two .mod files are identical.

# Description of Key Functions Used

1. The function that solves the model is:

   ```
   solve_one_constraint
   [zdatalinear zdatapiecewise zdatass oo_base M_base] =
   solve_one_constraint(modnam, modnamstar,
   constraint, constraint_relax,
   shockssequence, irfshock, nperiods, maxiter, init);
   ```

   Inputs:

   `modnam`: name of .mod file for the reference regime (excludes the .mod extension).

   `modnamstar`: name of .mod file for the alternative regime (excludes the .mod extension).

`constraint`: the constraint (see notes 1 and 2 below). When the condition in `constraint` evaluates to true, the solution switches from the reference to the alternative regime.

`constraint_relax`: when the condition in `constraint_relax` evaluates to true, the solution returns to the reference regime.

`shockssequence`: a sequence of unforeseen shocks under which one wants to solve the model (size T×nshocks).

`irfshock`: label for innovation for IRFs, from Dynare .mod file (one or more of the 'varexo').

`nperiods`: simulation horizon (can be longer than the sequence of shocks defined in `shockssequence`; must be long enough to ensure convergence back to the reference model at the end of the simulation horizon and may need to be varied depending on the sequence of shocks).

`maxiter`: maximum number of iterations allowed for the solution algorithm (20 if not specified).

`init:` the initial position for the vector of state variables, in deviation from steady state (if not specified, the default is steady state). The ordering follows the definition order in the .mod files.

Outputs:

`zdatalinear:` an array containing paths for all endogenous variables ignoring the occasionally binding constraint (the linear solution), in deviation from steady state. Each column is a variable, the order is the definition order in the .mod files.

`zdatapiecewise:` an array containing paths for all endogenous variables satisfying the occasionally binding constraint (the occbin/piecewise solution), in deviation from steady state. Each column is a variable, the order is the definition order in the .mod files.

`zdatass:` the steady state values of the variables. The ordering follows the definition order in the .mod files.

`oobase_, Mbase_` : structures produced by Dynare for the reference model – see Dynare User Guide.

2. The function that solves the model with two constraints is

```
[zdatalinear zdatapiecewise zdatass oo_00 M_00] =
 solve_two_constraints(modnam_00,modnam_10,modnam_01,modnam_11,...
 constraint1, constraint2,...
 constraint_relax1, constraint_relax2,...
 shockssequence,irfshock,nperiods,curb_retrench,maxiter,init);
```

Inputs:

`modnam_00`: name of the .mod file for reference regime (excludes the .mod extension).

`modnam_10`: name of the .mod file for the alternative regime governed by the first constraint.

`modnam_01`: name of the .mod file for the alternative regime governed by the second constraint.

`modnam_11`: name of the .mod file for the case in which both constraints force a switch to their alternative regimes.

`constraint1`: the first constraint (see notes 1 and 2 below). If `constraint1` evaluates to true, then the solution switches to the alternative regime for condition 1. In that case, if `constraint2` (described below) evaluates to false, then the model solution switches to enforcing the conditions for an equilibrium in `modnam_10`. Otherwise, if `constraint2` also evaluates to true, then the model solution switches to enforcing the conditions for an equilibrium in `modnam_11`.

`constraint_relax1`: when the condition in `constraint_relax1` evaluates to true, the solution returns to the reference regime for `constraint1`.

`constraint2`: the second constraint (see notes 1 and 2 below).

`constraint_relax2`: when the condition in `constraint_relax2` evaluates to true, the solution returns to the reference regime for `constraint2`.

`shockssequence`: a sequence of unforeseen shocks under which one wants to solve the model

`irfshock`: label for innovation for IRFs, from Dynare .mod file (one or more of the 'varexo')

`nperiods`: simulation horizon (can be longer than the sequence of shocks defined in `shockssequence`; must be long enough to ensure convergence back to the reference model at the end of the simulation horizon and may need to be varied depending on the sequence of shocks).

`curb_retrench:` a scalar equal to 0 or 1. Default is 0. When set to 0, it updates the guess based of regimes based on the previous iteration. When set to 1, it updates in a manner similar to a Gauss-Jacobi scheme, slowing the iterations down by updating the guess of regimes only one period at a time.

`maxiter`: maximum number of iterations allowed for the solution algorithm (20 if not specified).

`init:` the initial position for the vector of state variables, in deviation from steady state (if not specified, the default is a vector of zero implying that the initial conditions coincide with the steady state). The ordering follows the definition order in the .mod files.

Outputs:

`zdatalinear:` an array containing paths for all endogenous variables ignoring the occasionally binding constraint (the linear solution), in deviation from steady state. Each column is a variable, the order is the definition order in the .mod files.

`zdatapiecewise:` an array containing paths for all endogenous variables satisfying the occasionally binding constraint (the occbin/piecewise solution), in deviation from steady state. Each column is a variable, the order is the definition order in the .mod files.

`zdatass:` a vector that holds the steady state values of the endogenous variables ( following the definition order in the .mod file).

`oo00_, M00_` : structures produced by Dynare for the reference model – see Dynare User Guide.

3. The functions `solve_one_constraint` and `solve_two_constraints` assume that the endogenous, exogenous variables and the model parameters are declared in exactly the same order in all .mod files.

   In general, to create the additional files, it pays to simply make a replica of the reference .mod file and amend the relevant equations.

4. The only **restrictions for the .mod files is that they may accommodate at most one lag and one lead** of the endogenous variables. The conditions that govern the switching across the reference and alternative(s) model(s) may only involve contemporaneous variables. With appropriate redefinitions, these restrictions come at no loss of generality.

# Additional Notes

1. **Writing the constraint**

   **Model with irreversible capital.** The original occasionally binding constraint in the model is

   $$i_t \geq \log\left(\phi \cdot I_{ss}\right)$$

   where $i_t$ is natural logarithm of investment (see mod file), $I_{ss}$ is steady state investment in levels, and $\phi$ is a parameter.

   For the constraint to be violated, in the candidate solution calculated under the assumption that the constraint does not bind the following must be true

   $$i_t < \log\left(\phi \cdot I_{ss}\right)$$

   Rewrite each variable as

   $$x_t \equiv \widetilde{x}_t + x_{ss}$$

In the runsim_irrcap.m code, the constraint will have to be expressed in linearized form as

$$\widetilde{i}_t + i_{ss} < \log(\phi \cdot I_{ss}) \Longleftrightarrow \widetilde{i}_t < \log(\phi)$$

and the string `constraint` will be

$$i < log(PHII)$$

Therefore note that in the mod file, $i$ will denote the variable, whereas in the `constraint` the same $i$ will denote the variable minus its steady state level.

**Model with borrowing constraint.** The occasionally binding constraint specifies that $B_t = mY_t$, which implies that $\lambda_t > 0$. This constraint is violated if in the candidate solution the following is true

$$\lambda_t < 0$$

Rewrite $\lambda_t$ as

$$\lambda_t \equiv \widetilde{\lambda}_t + \lambda_{ss}$$

In the runsim_borrcon.m file, the constraint will be expressed in linearized form as

$$\widetilde{\lambda}_t + \lambda_{ss} < 0 \Longleftrightarrow \widetilde{\lambda}_t < -\lambda_{ss}.$$

and the string `constraint` will be

$$lambda < -lambda\_ss$$

To write `constraint_relax`, note that the constraint will then bind again if at any point borrowing exceeds its upper bound. That is

$$\widetilde{B}_t + B_{ss} > m\left(\widetilde{Y}_t + Y_{ss}\right) \Longleftrightarrow \widetilde{B}_t > m\widetilde{Y}_t,$$

and the string `constraint_relax` will be

$$B > mY$$

2. **In the toolbox, `constraint` and `constraint_relax` only admit contemporaneous endogenous variables**. Note that there is no loss of generality since appropriate redefinitions can accommodate a general lead and lag structure.

3. In all runsim_*.m files, we declare M_ and oo_ to be global variables (for use by Dynare)

4. One does not need to specify the steady state of any alternative model. All models are approximated around the steady state of the reference model (which needs to satisfy the Blanchard-Kahn conditions). Since local and global stability need not coincide, the conditions for an equilibrium in the alternative regime(s) need not satisfy the BK conditions.

5. Values for the parameters (whether in the .mod or in external files) are specified only for the reference model. Parameter values specified for the alternative model(s) are ignored by the code (but not the list of parameters, which is used for an error check). If a parameter only enters the alternative model, it needs to be declared as parameter and assigned a value in the reference model.

6. We have strived to minimize possible conflicts between local variables used in coding the solution algorithm and parameter names declared in the .mod files. We have reserved variable names with a trailing underscore – i.e., names such as "$myfavoritevariablename\_$" – for the solution routines. Variable names with trailing underscores should be avoided in the .mod files.