COPPs toolkit manual

Oliver de Groot[*] Falk Mazelis[†] Roberto Motto[‡] Annukka Ristiniemi[§]

Version 1.0: March 2021

COPPs toolkit repository: https://github.com/COPPsToolkit/COPPs

Please cite the COPPs toolkit if you use it in your research by citing the following paper:

de Groot, O., F. Mazelis, R. Motto, A. Ristiniemi (2021), "A Toolkit for Computing Constrained Optimal Policy Projections (COPPs)", Forthcoming ECB working paper

BibTeX file:

```
@article{deGrootetal2021toolkit,
author = {de Groot, Oliver and Mazelis, Falk and Motto, Roberto and Ristiniemi, Annukka},
title = {A Toolkit for Computing Constrained Optimal Policy Projections (COPPs)},
year = 2021,
journal = {Forthcoming ECB working paper}}
```

If you spot any bugs in the toolkit, please email `falk.mazelis@ecb.europa.eu`

[*]University of Liverpool Management School & CEPR, Liverpool, UK (oliverdegroot@gmail.com)
[†]European Central Bank, Frankfurt, Germany (falk.mazelis@ecb.europa.eu)
[‡]European Central Bank, Frankfurt, Germany (roberto.motto@ecb.europa.eu)
[§]European Central Bank, Frankfurt, Germany (annukka.ristiniemi@ecb.europa.eu)

Contents

1 Introduction

This toolkit can compute optimal policy projections under commitment, discretion, or limited-time commitment.[1] The toolkit can handle two policy instruments and can impose constraints on policy such as lower or upper bounds. The toolkit also has options to mitigate the effect of the forward guidance puzzle. Finally, the toolkit is compatible with Dynare (Adjemian et al., 2011).[2] Many example files are included with the toolkit.

The toolkit requires two basic inputs:

1. **A model and a baseline projection**: The model needs to be written as a `Dynare .mod` file and the baseline projection needs to be stored as an `Excel .xlsx` file. The former will be used to calculate impulse response functions (IRFs) to anticipated policy shocks. You will need to make a few small modifications to your `.mod` file to make it compatible with the toolkit. These modifications are listed in Section 2.
2. **The policymaker's objective function and policy instruments**: These are set as parameters and are explained in Section 3.

The main toolkit file is a `Matlab` file called `Run_.m`. The toolkit contains many examples. RunCOPPs.m in the main folder can be used as a template for a new project. In this file you must specify the required parameters (see Section 3.1) and any optional parameters (see Section 3.2). The file then calls the following function to compute the optimal policy projections:

```
projections = SolveCOPPs(params);
```

and the following function to generate the plots:

```
PlotCOPPs(projections,params);
```

The plotting parameters are described in Section 3.3.

[1]The methodology is explained in detail in de Groot, Mazelis, Motto and Ristiniemi (2021).
[2]The toolbox has been tested using Dynare version 4.5.7

2 Adding a model

Any model written as a `Dynare .mod` file, whether calibrated or estimated, can be added to the toolkit. Some commands have to be added to the `.mod` file in order for it to be read into the toolbox. The additional codes are written inside Macro commands.[3]

Data: The toolkit computes policy instrument projections that minimize the policymaker's loss function (possibly subject to constraints) relative to a 'baseline'. This baseline is incorporated into the toolkit from a separate `datafile` saved in the same folder as the model. In the ideal case, the data includes projections for the target and instrument variables until the period when all shocks have ceased to affect the economy and the variables have returned to their steady states. If the horizon of the baseline projection data is too short, or there are variables missing, as is typical in practice, the Kalman smoother in `Dynare` can be used to complete the baseline projection. It is therefore possible to rely on data that does not feature the entire projection horizon, a return to steady state, or even the full complement of instruments and target variables. However, the toolkit will always use the baseline and construct the optimal policy projection relative to this baseline. Thus, if the baseline features implausible dynamics, this will directly affect the optimal policy projections.

To ensure a plausible baseline, it is essential to ensure a correct mapping between the data and the model. The toolkit relies on the `Dynare` infrastructure for importing data into the model. All conventions specific to `Dynare` therefore need to be satisfied for the optimal policy projections to function well. For a general guide to specifying observation equations in `Dynare`, the reader is referred to Pfeifer (2013). Here we simply demonstrate good practice using the Smets and Wouters (2007) example from de Groot et al. (2021).

In this example, we specify the short-term rate as the policy instrument. We therefore define the model variable `r` as the first instrument in the run-file of the toolkit (see Section 3.1 for the syntax). Interest rate data is commonly defined in annual terms, but the model variable is defined in quarterly terms. Following Smets and Wouters (2007), we therefore construct an appropriately transformed time-series in the `datafile` called `robs`, which is based on the effective Federal Funds Rate (detailed in the sheet 'MyData' in `usmodel_data.xls`, saved in the folder `\Models\SW07`). Note that this transformation could alternatively be applied in the model file directly. We next apply a model-based transformation: the model defines variables in deviations from their steady state. To ensure that the policy rate returns to a long-run value that is different from zero, we need to define the steady state of `robs`. The steady state value of the quarterly short-term rate is provided by the parameter `conster` and the calculation based on structural parameters can be found in Smets and Wouters (2007). The observation equations that maps the short-term rate to the model is then `robs = r +`

[3] See `Dynare` manual for more information.

conster, which is included in the .mod file.

As an example of a target variable, consider annual inflation in SW07. We assume that the long-run value of inflation is 2%, in line with the Fed's Statement on Longer-Run Goals and Monetary Policy Strategy. Since the model includes quarterly inflation in the Phillips Curve (pinf), the data we incorporate are defined in quarterly terms (pinfobs), and we assume a quarterly steady state of constepinf $= 0.5\%$ for inflation (as opposed to estimating the steady state value as done by SW07). The observation equation then links the data to the model equation via the assumed steady state: `pinfobs = pinf + constepinf`. To adjust the aim of the policymaker to a different value, we would adjust the steady state value constepinf.

Smoother: The COPPs Toolkit runs calib_smoother in Dynare to produce the baseline around which optimal policy is formulated. You need to provide a data file and an appropriate mapping between data and model (using the varobs command in Dynare). The data file can include both historical values and forecasts. The periods that are to be treated as historical will be set in the COPPs code (and detailed in Section 3.1 below).

To construct any remaining variables, the model filters the shocks that explain the data. The forecast in the data file can be provided up to any horizon, or left blank, in which case the Kalman smoother produces a forecast assuming no future shocks and a return to the steady state. Unless the .mod file already includes the calib_smoother command with reference to a data file, it should be added as in the below example (see, e.g., Models\SW07\Smets_Wouters_2007_GB09.mod).[4]

```
@#ifdef SMOOTHER
    calib_smoother(datafile=usmodel_data);
@#endif
```

Required parameters: Two parameters need to be provided numerically, or should be created inside the .mod file: the planner discount factor and a steady state of the nominal interest rate (see Section 3.1 and Section 3.2, respectively). The planner discount factor is required for the calculation of optimal policy. The steady state nominal interest rate is required when implementing a lower bound on the policy rate.

In the Smets and Wouters (2007) model the steady state annual interest rate and discount rate are derived from other parameters. The calculation can be done in the .mod file by adding the code:

```
beta_ss = 1/(1+constebeta/100);
ss_r_ann = (((1+constepinf/100)/((1/(1+constebeta/100))*(1+ctrend/100)^(-csigma)))-1)*100*4;
```

[4]The calib_smoother command may be inside a Macro directive called "SMOOTHER" to ensure it is only called when constructing the baseline.

3 COPPs Parameters

3.1 Required parameters

Required parameters need to be set by the user. There is no default value.

ModelDirectory = PATH

ModelDirectory specifies the folder where the mod file is located relative to the main folder of the toolkit.

Example: `params.ModelDirectory = 'Models\SW07';`

ModelFilename = FILE_NAME

ModelFilename specifies the name of the mod file that contains the underlying model. This file needs to be saved in the ModelDirectory.

Example: `params.ModelFilename = 'Smets_Wouters_2007_GB09_mod210206';`

PlannerDiscountFactor = PARAMETER_NAME | DOUBLE

PlannerDiscountFactor specifies the discount factor β of the policymaker as defined in Equation (1), Subsection 2.1 of the paper. The discount factor may be provided as a parameter name as defined in the model, in which case the value will be pulled directly from the mod file. Alternatively, it can be provided as a numerical value.

Example: `params.PlannerDiscountFactor = 'beta_ss';`

Example: `params.PlannerDiscountFactor = 0.99;`

LossFunctionVariables = VARIABLE_NAMES

LossFunctionVariables specifies the names of the variables that enter the policymakers loss function as defined in Equation (1), Subsection 2.1 of the paper. These variables need to be specified in the mod file.

Example: `params.LossFunctionVariables = {'obs_pinf_4q';'ygap';'dr'};`

PolicyInstrumentsAndShocks = VARIABLE_NAMES

PolicyInstrumentsAndShocks specifies the names of the policymakers' instruments as defined in Equation (3), Subsection 2.3 of the paper. These variables need to be specified in the mod file. The number of instruments needs to be at least one, with the policy rate in quarterly terms as the first instrument. The current version of the toolkit limits the number of instruments to two.

Example: `params.PolicyInstrumentsAndShocks = ‘r’ ,‘em’;`

PastPeriods = INTEGER

PastPeriods specifies the number of periods in the dataset that are taken to be historical. The optimal policy projections are provided for the periods following PastPeriods and extend for T periods (see optional parameter T below).

Example: `params.PastPeriods = 219;`

3.2 Optional parameters

Optional parameters may be set by the user. In case there is no user-specified option, the parameter will take on a default value.

3.2.1 Optimal policy commands

`Dynare`Options = STRING

`Dynare`Options specifies the type of optimal policy, see Subsection 2.4 of the paper.

Default: none.

Example: `params.DynareOptions = ‘ -DFlatPC’;` to provide macro command to the mod file (see `Dynare` macro processing language).

PolicyType = ‘COM’ | ‘DIS’

PolicyType specifies the type of optimal policy, see Subsection 2.4 of the paper.

Default: `‘COM’`

Example: `params.PolicyType = ‘COM’;`

LossFunctionWeights = DOUBLE

LossFunctionWeights provides the preference of the policymakers with respect to the relative weight of each loss function variable as defined in Equation (1), Subsection 2.1 of the paper.

Default: equal weighting of all loss function variables.

Example: `params.LossFunctionWeights = [1,0.25,4];`

T_loss = INTEGER

Length of the projection over which the loss is calculated. Must be sufficiently long such that all the policy preference and policy instrument variables have converged back to the steady state.

Default: `40`

Example: `params.T_loss = 40;`

H_policy = INTEGER

Maximum horizon for policy announcements. May not exceed T_loss.

Default: length of *T_loss*.

Example: `params.H_policy = 40;`

T_full = INTEGER

Entire length of the projection. Smoothed for all variable projections that are not provided by the user. Needs to be larger than T_loss.

Default: `100`

Example: `params.T_full = 100;`

3.2.2 Occasionally binding constraints on policy instruments

Constraint = 'ON' | 'OFF'

Controls the enforcement of instrument bounds and corridors.

Example: `params.Constraint = 'OFF';`

RateMin = DOUBLE

Effective lower bound of the policy rate relative to the steady state value, in annual terms. Defined as $\bar{i}$ in Equation (14), Example 7.1 of the paper. params.PolicyRateStSt needs to be provided.

Default: `0`

Example: `params.RateMin = 0;`

PolicyRateStSt = PARAMETER_NAME | DOUBLE

PolicyRateStSt specifies the steady state value of the nominal policy rate in annual terms. The steady state of the nominal policy rate may be provided as a parameter name as defined in the model, in which case the

value will be pulled directly from the mod file. Alternatively, it can be provided as a numerical value.
Default: none.
Example: `params.PolicyRateStSt = 'ss_r_ann';`
Example: `params.PolicyRateStSt = 4;`

RateDevMin = DOUBLE

Allows for the possibility that the policymaker is reluctant to make large policy changes from the baseline path. Thus, policy variables are constrained to lie within a corridor of the baseline. See Example 7.3 and Figure 7 of the paper. `RateDevMin` specifies the time-varying lower limit of the corridor around the baseline path of the first policy instrument (assumed to be the policy rate). Defined in units of the first policy variable as provided in `PolicyInstrumentsAndShock`. May not exceed `T_loss`.
Default: none.
Example: `params.RateDevMin = [0.1 0.1 0.1 0.1];` in the first four periods up to 10bps negative deviations of the quarterly policy rate from the baseline path, which is defined in percentage points. Translates into 40bps in the annual policy rate.

RateDevMax = DOUBLE

`RateDevMax` specifies the time-varying upper limit of the corridor around the baseline path. Can be used to implement date-based FG, see Example 7.2. May not exceed `T_loss`.
Default: none.
Example: `params.RateDevMax = .1;`

QeMin = DOUBLE

Lower bound of balance sheet relative to the steady state value. Steady state is assumed to be 0. Defined in units of the second policy variable as provided in `PolicyInstrumentsAndShock`.
Default: `0`
Example: `params.QeMin = 0;`

QeMax = DOUBLE

Lower bound of balance sheet relative to the steady state value. Steady state is assumed to be 0. Defined as $\bar{q}$ in Equation (14), Example 7.1 of the paper. Defined in units of the second policy variable as provided in `PolicyInstrumentsAndShock`.

Default: `100`

Example: `params.QeMax = 100;`

QeDevMin = DOUBLE

Allows for the possibility that the policymaker is reluctant to make large policy changes from the baseline path. Thus, policy variables are constrained to lie within a corridor of the baseline. See Example 7.3 and Figure 7 of the paper. `QeDevMin` specifies the time-varying lower limit of the corridor around the baseline path of the second policy instrument (assumed to be asset purchases). Defined in units of the second policy variable as provided in `PolicyInstrumentsAndShock`. May not exceed `T_loss`.

Default: none.

Example: `params.QeDevMin = [3 3 3 3];` in the first four periods up to 3p.p. negative deviations of the balance sheet from the baseline path, which is defined in percentage points.

QeDevMax = DOUBLE

`QeDevMax` specifies the time-varying upper limit of the corridor around the baseline path. May not exceed `T_loss`.

Default: none.

Example: `params.RateDevMax = 3;`

3.2.3 Attention dampening

TYPE = 'I' | 'II' | 'III' | 'IV'

Type of attention dampening to policy announcements. I: (constant) inattention; II: Credibility (decaying attention); III: Finite planning horizon; IV: Learning. For details of the different types, their parameterizations and effects, see de Groot and Mazelis (2020).

Default: `'I'`

Example: `params.TYPE = 'I';`

alpha = DOUBLE

Degree of attention to policy announcements for Types I and II. Under inattention, a fraction $1 - \alpha$ of agents are inattentive to optimal policy announcements of the policymaker. Under credibility, a decaying fraction of agents believe optimal policy announcements at further horizons. In particular, a fraction α believe announcements 1-quarter ahead, a fraction α^2 believe announcements 2-quarters ahead etc. $\alpha \in [0; 1]$. The bounds are given by 1 = Full attention/credibility, and 0 = Complete inattention/incredibility.

Default: `0.7`

Example: `params.alpha = 0.7;`

N_planning = INTEGER

Number of periods of the planning horizon for Type III. N_planning $\in [0; T\_loss]$. The bounds are given by T_loss = Full attention, and 0 = Complete inattention.

Default: `4`

Example: `params.N_planning = 4;`

beta1 = INTEGER

Under learning, a large fraction of agents initially dismiss optimal policy announcements, but this fraction falls as time passes and the policymaker is able to show its commitment to its promises. The learning scenario is parameterized by a 2-parameter logistic function where beta1 controls the speed of learning, while beta2 controls the initial beliefs. Number of periods of the planning horizon. $\beta_1 \in [0; T\_loss]$.

Default: `1`

Example: `params.beta1 = 1;`

beta2 = INTEGER

The learning scenario is parameterized by a 2-parameter logistic function where beta1 controls the speed of learning, while beta2 controls the initial beliefs. $\beta_2 \in [0; T\_loss]$.

Default: `5`

Example: `params.beta2 = 5;`

3.2.4 Solution algorithm

MaxIter = INTEGER

`MaxIter` specifies the maximum number of iterations for solving Optimal Commitment.

Default: 1e4.

Example: `params.MaxIter = 1e4;`

Crit = DOUBLE

`Crit` specifies the minimum improvement required to continue iterating over `quadprog` for solving Optimal Commitment.

Default: 1e-3.

Example: `params.Crit = 1e-3;`

Update = DOUBLE

`Update` specifies the speed at which to update initial conditions for the next iteration in `quadprog` for solving Optimal Commitment.

Default: 0.1.

Example: `params.Update = 0.1;`

3.3 Plotting parameters

plotting.NoPlot = INTEGER

`NoPlot` suppresses the graphical output if set to '1'.

Default: 0.

Example: `params.plotting.NoPlot = 1;`

plotting.VarsToPlot = CELL ARRAY

`VarsToPlot` specifies the variables to be plotted, as well as the vertical axis limits and title of each variable.

Default: Target and instrument variables.

Example:

```
params.plotting.VarsToPlot = {...
    'obs_pinf_4q'  , [ -1 ,  4  ]  , 'Inflation (annual, P.P.)'     ; ...
    'ygap'         , [ -9 ,  6  ]  , 'Output gap (P.P.)'            ; ...
    'obs_r_ann'    , [ -8 ,  7  ]  , 'Interest rate (annual, P.P.)' ; ...
    'qeobs'        , [ -5 ,  50 ]  , 'Asset holdings (% of GDP) '   ; ...
};
```

plotting.Data = CELL ARRAY

`Data` specifies the projections to be plotted, as well as their graphical depiction and legend entry. `simple_rules` refers to the user-provided data and timeseries filtered via the provided model with the simple policy rules. `COPPs` refers to the timeseries constructed via the (constrained) optimal policy. The second argument sets the line style. The third argument sets the line width. The fourth argument sets the line color. The last argument sets the legend entry.

Default: The example depicts the default values.

Example:

```
Data = {projections.simple_rules.data    , '-'  , 1 ,[0,0,0], 'Baseline' ; ...
        projections.COPPs.data           , '--' , 1 ,[0,0,1], 'COPPs'    ; ...
        };
```

plotting.first_date = DATETIME OBJECT

The first date in the sample is used for creating a date series for plotting. Set a year, month, and date inside the datetime function. If first_date is not defined, integers are automatically plotted on the horizontal axis such that 0 corresponds to the last data point.

Example with 1st of April 2020: `params.plotting.first_date = datetime(2020,4,1)`

plotting.freq = 'month' | 'quarter' | 'year'

The data frequency is used for creating a date series for plotting. The freq can be "month", "quarter", or "year".

plotting.P_past = INTEGER

`P_past` sets the number of periods prior to the start of optimal policy that are shown in the plot.

Default: 20.

Example: `params.plotting.P_past = 10;`

plotting.P_future = INTEGER

`P_future` sets the number of periods from the start of optimal policy that are shown in the plot.

Default: 36.

Example: `params.plotting.P_future = 40;`

References

Adjemian, Stéphane, Houtan Bastani, Michel Juillard, Fréderic Karamé, Junior Maih, Ferhat Mihoubi, George Perendia, Johannes Pfeifer, Marco Ratto, and Sébastien Villemot (2011) 'Dynare: Reference manual version 4.' Dynare Working Papers 1, CEPREMAP

de Groot, Oliver, and Falk Mazelis (2020) 'Mitigating the forward guidance puzzle: inattention, credibility, finite planning horizons and learning.' *ECB working paper 2426*

de Groot, Oliver, Falk Mazelis, Roberto Motto, and Annukka Ristiniemi (2021) 'A Toolkit for Computing Constrained Optimal Policy Projections (COPPs).' *ECB working paper*

Pfeifer, Johannes (2013) 'A guide to specifying observation equations for the estimation of DSGE models.' *Unpublished manuscript*

Smets, Frank, and Rafael Wouters (2007) 'Shocks and Frictions in US Business Cycles: A Bayesian DSGE Approach.' *American Economic Review* 97(3), 586–606