hetlearn: technical appendix

Heterogeneous expectations with adaptive learning and switching in IRIS models

Tolga Özden*

Version of 23 October 2026. Tested with IRIS 20230622 and MATLAB R2025a.

Abstract

This appendix documents the mathematics, the assumptions and every user-facing option of the HetLearn toolbox. The toolbox takes a linear IRIS model (nonlinear models have a code path but no test) with forward-looking terms and replaces each one-step expectation by a population mix of rational (RE) agents and adaptive learners whose shares switch on past forecast performance, as in Özden (2025). It writes the extended model file, runs a history stage (learning on the RE smoother, the filtered RE states, an inversion of the learning model, or a time-varying Kalman filter), and then forecasts quarter by quarter with the beliefs and shares in force, under a projection facility. On top of the forecast stage sit conditional forecasts, Ramsey policy, density forecasts, shock decompositions and a recursive real-time forecast evaluation. Each option is given with its equations, the assumptions behind it, its default, its allowed values, its interactions with other options and a short configuration example. Each equation names the file that implements it.

Contents

1 Introduction **5**

1.1 What the toolbox does . . . 5
1.2 Public functions . . . 5
1.3 Scope of this appendix . . . 6
1.4 How an option is documented . . . 6
1.5 Option precedence and merging . . . 7
1.6 Conventions . . . 7

2 Notation and timing **7**

2.1 Symbols . . . 7
2.2 The sequence of one quarter . . . 9
2.3 Information sets . . . 9

3 Model requirements and the model package **9**

3.1 What a model must have . . . 9

*Code and documentation: `https://github.com/HetLearn/iris_toolbox_learning`, released under the MIT License.

3.2 The package contract: `model_spec.m` . . . 10
3.3 Package options . . . 11

4 The extended model **11**
4.1 Lead detection . . . 12
4.2 Rewriting and the expectation block . . . 12
4.3 Model-file toggles . . . 13
4.4 Parameters of the extended model . . . 14
4.5 Generated files and the cache . . . 14
4.6 The collapse test . . . 15
4.7 Generator call options . . . 15

5 Expectations: the aggregate and the modes **15**
5.1 Options . . . 15

6 RE agents **16**
6.1 Model-consistent RE agents . . . 16
6.2 Anchored RE agents . . . 17
6.3 Existence and models without leads . . . 17
6.4 Option . . . 18

7 Information timing **18**
7.1 The learners' information set . . . 18
7.2 The RE agents' information set . . . 19

8 Perceived laws of motion **21**
8.1 The PLM equation . . . 21
8.2 Options . . . 21

9 Belief updating **24**
9.1 Recursive least squares . . . 24
9.2 Gain rules . . . 25
9.3 The decreasing gain . . . 26
9.4 The endogenous gain of Kostyshyna, Özden and Zhang . . . 27

10 Initial beliefs **30**
10.1 Options . . . 30

11 Switching between RE and learning **32**
11.1 Forecast errors, fitness and shares . . . 32
11.2 Options . . . 33

12 The projection facility **36**
12.1 PLM blocks and their roots . . . 36
12.2 The three settings . . . 36
12.3 Kept shares . . . 37
12.4 Where the check runs . . . 37
12.5 Events . . . 38

13 The history stage **38**
13.1 The RE filter and smoother . . . 39
13.2 Learning on the RE history . . . 40
13.3 The inversion history . . . 42
13.4 The Kalman history . . . 43

14 The forecast stage **45**
14.1 The quarter-by-quarter loop . . . 45
14.2 Origin and horizon . . . 45
14.3 Shocks over the horizon . . . 46
14.4 Conditions: exogenize and endogenize . . . 47
14.5 The one-period step . . . 49
14.6 Identification and comparisons . . . 49

15 Monetary policy: rules and Ramsey **50**
15.1 The policy section of a package . . . 50
15.2 The planner's problem . . . 51
15.3 Rule-only names . . . 51
15.4 Loss accounting . . . 51
15.5 Options . . . 51

16 Density forecasts **53**
16.1 Options . . . 54

17 Historical shock decompositions **58**
17.1 Call options . . . 59

18 Recursive forecast evaluation **60**
18.1 Call options . . . 61

19 Scenario files and caches **61**
19.1 Scenario files . . . 61
19.2 Caches . . . 62

20 Outputs and the result struct **62**

20.1 The result struct 62
20.2 Learning fields 62
20.3 Share names 63
20.4 Files in `results/<id>/` 63

21 Validation and the test suite **63**
21.1 Checks inside every run 63
21.2 The test suite 64
21.3 Known limitations 64

A Option index **64**

1 Introduction

1.1 What the toolbox does

HetLearn is a MATLAB package (`toolbox/+hetlearn/`) built on the IRIS toolbox (IRIS Solutions Team, 2023). A user supplies a model package: an IRIS model file, a parameter function, an IRIS-readable database, the observables and a filter range. The toolbox

1. rewrites the model so that every lead x_{t+k} of a transition variable becomes an explicit expectation variable, and gives every forward-looking variable an expectation block that mixes an RE forecast and a learners' forecast (Section 4);
2. runs an RE Kalman filter and smoother over the data (Section 13);
3. lets the learners form beliefs over the history, by recursive least squares (RLS) on a perceived law of motion (PLM), the reduced-form forecasting rule they estimate, while the population shares follow a discrete-choice rule on past squared forecast errors (Sections 8–11);
4. forecasts one quarter at a time from a forecast origin: in each quarter the model is solved with the beliefs and shares in force and simulated for that quarter only, and the beliefs and shares are updated with the new data point (Section 14).

The learning and switching block follows Özden (2025), equations (12)–(16) and Appendix D of that paper. The endogenous gain follows Kostyshyna, Özden, and Zhang (2024, eq. 9), hereafter KOZ, who apply the adaptive step-size algorithm of Kostyshyna (2012); the gain rule `'koz'` is named after them.

1.2 Public functions

The toolbox is used through these functions; `help hetlearn.<name>` gives the full text of each.

Call	Purpose
`res = hetlearn.run(cfg, ...)`	one scenario; options `'save'`, `'results_dir'`, `'quiet'`, `'model'` (preloaded), `'filter'` (an RE filter output)
`T = hetlearn.run_files(names, ...)`	runs scenario functions in order; a failure is reported and the next one runs
`T = hetlearn.compare(ids, vars, range)`	levels `<var>__<id>` and differences `<var>__<id>_minus_<id1>` of saved runs
`D = hetlearn.decompose(res, vars, range, opts)`	Section 17
`E = hetlearn.evaluate(name, variants, origins, opts)`	Section 18
`T = hetlearn.evaluate_tables(errors, opts)`	tables from saved errors
`o = hetlearn.defaults()`	the option tree
`opts = hetlearn.load_options(cfg, master)`	merge and validate
`spec = hetlearn.load_spec(name)`	package contract
`[d, file] = hetlearn.load_database(spec)`	the package database
`[m, info] = hetlearn.load_model(spec, opts)`	generate, load, assign, solve the extended model

Call	Purpose
`H = hetlearn.re_filter(spec, opts, ...)`	RE filter and smoother; `'force'`, `'end'`, `'model'`
`meta = hetlearn.extend_model(spec, plm, ...)`	the generator
`[A, info] = hetlearn.re_anchor_coefficients(spec, P, ...)`	anchored coefficients
`t = hetlearn.find_peak(db, name, from, to)`	date of a series' maximum in a window

1.3 Scope of this appendix

This appendix covers the core toolbox: two agent types (RE agents and learners), adaptive learning, switching and every stage built on them. It leaves out two parts of the repository:

- the bounded-rationality extensions, which Section 7 of the user guide describes as the further expectation rules: the options `expectations.types`, `expectations.transforms`, `shares.init`, `shares.fixed`, the type kinds `'rule'` and `'re_mean'`, and the N-type paths of the engine (`+engine/ntypes.m`). Every one of these options is off by default, and a run that sets none of them takes the two-type code described here;
- the Bayesian estimation module (`estimation/`, package `+hlest`), which has its own copy of the toolbox and no scenario uses it.

The time-varying Kalman history (`history.mode = 'kalman'`) belongs to the extensions but works with two types, so Section 13.4 documents its two-type form.

1.4 How an option is documented

Each option has its own entry with the same fields:

Default the value in `hetlearn.defaults()` (`defaults.m`). The project baseline in `config/master_options.m` overrides some defaults for the SW07 scenarios; those values are noted where they differ, but the entries document the toolbox defaults.

Values the values `hetlearn.load_options` accepts (`load_options.m`).

Math the equations the option selects or parameterizes.

Assumptions what the option takes as given about agents, information or the model.

Interactions other options that change its effect, combinations that are errors, and options it makes irrelevant.

Code the files that implement it. Paths are relative to `toolbox/+hetlearn/`; `+engine/forecast.m` is the function `hetlearn.engine.forecast`.

Example one or two lines of a scenario's `cfg`.

Appendix A lists every option with its default and the section that documents it.

1.5 Option precedence and merging

`opts = hetlearn.load_options(cfg, master)` (`load_options.m`) builds the option tree in three layers: the defaults of `hetlearn.defaults()`, then `config/master_options.m`, then the scenario's `cfg`, each layer overriding the previous one field by field. `master_options()` is found on the path or in `<root>/config/`; the optional second argument replaces it (the tests use their own `tests/scenarios/master_options.m`). Rules:

- the merge is recursive; a field that does not exist in the defaults is `hetlearn:options:unknown`, naming the full path (typos are caught);
- *replaced as a whole*: `conditions` and `policy.loss` (a loss struct without `dr` gives `dr` weight 0);
- *free-form*, merged without name checks: `beliefs.init.values`, `beliefs.plm.by_var`, `beliefs.plm.derived` (must hold strings), `gain.by_var`, `shares.fixed_n2` (as a struct), `shocks.add`, `report`;
- strings become char; a char where a cellstr is expected becomes a one-element cellstr (`expectations.variables` other than `'all'`, `shares.master`, `compare_with`, `shocks.exclude`, `density.shocks`, `history.inversion.shocks`, `gain.koz.master`, `report.variables`);
- the removed options `projection.max_root`, `.method`, `.scope`, `.on_failure`, `switching.error_scale`, `.error_scale_ref` and the removed bounded-rationality options raise `hetlearn:options:removed`, which names the replacement. A saved `res.cfg` that still holds a removed field at its old default value loads without error.

Validation errors: `hetlearn:options:type` (wrong type), `:value` (outside the enumeration or range, or a forbidden combination), `:missing` (a required companion option), `:unsupported` (combinations not derived or built), `:notImplemented` (`history.mode = 'hetero_filter'`). Checks that need the model (names of variables, shocks and targets) are made later by the generator, the PLM builder and the engine.

1.6 Conventions

- Quarters are IRIS dates. The forecast origin T_0 is the last quarter of data used; the horizon is $T_0 + 1, \ldots, T_0 + H$.
- Model units are those of the package. In the Smets and Wouters (2007) model (SW07), the main package of the repository, and in the toy model of the tests (`tests/toy_nk/`), inflation and the policy rate are in percent per quarter; reports multiply by `spec.report.scale` (4 annualizes). Learning series are always in model units.
- The SW07 data, seven US observables over the filter range 1965Q1–2026Q2 with every observable missing in 2020Q1–2020Q3, are described in `models/sw07/README.md`.
- This appendix calls the adaptive agents learners; the generated names use `BR` (bounded rationality) for them. Share names: n_1 is the RE share, $n_2 = 1 - n_1$ the learners' share. Options give the learners' share (`shares.init_n2`, `shares.fixed_n2`); Section 20.3 lists every share name.
- Tolerances quoted in this appendix are the ones the code checks.

2 Notation and timing

2.1 Symbols

Symbol	Meaning	Name in the code
t, T_0, H	quarter, forecast origin, horizon	`forecast.origin`, `forecast.horizon`
$\mathcal{F}$	forward-looking transition variables of the model	`meta.forward_vars`
$\mathcal{L} \subseteq \mathcal{F}$	learning set: variables whose expectation is the RE/learner mix	`expectations.variables`, `info.learnVars`
$x \in \mathcal{L}$	a learning variable	`x`
$\tilde{E}_t x_{t+1}$	aggregate one-step expectation of x in the model	`x_exp1`
$f_t^{\mathrm{RE}}(x)$	RE agents' forecast of x_{t+1} made in t	`x_exp1_RE`
$f_t^{\mathrm{BR}}(x)$	learners' forecast of x_{t+1} made in t	`x_exp1_BR`
$n_{1,t}$, $n_{2,t}$	discrete-choice RE and learner shares computed at the end of t	`res.learning.<x>.n1`, `n2`
$n_{1,t}^m$	RE share in the model in quarter t	parameter `n1_x`; `n1_model`
ϕ_t	PLM coefficients at the end of t (in force in $t+1$)	`S.phi.<target>`
R_t	RLS second-moment matrix	`S.R.<target>`
z_{t-1}	regressor vector of a PLM equation, entries dated $t-1$ or earlier	`Z` in `update_learning.m`
g_t	gain of the RLS update at the end of t	`res.learning.<target>.gain`
γ_0	constant gain	`gain.value`
ζ_t^{RE}, ζ_t^{BR}	fitness of the two predictors	`zetaRE`, `zetaBR`
χ, ω	intensity of choice, memory	`switching.intensity`, `switching.memory`
FE_t^i	squared one-step forecast error of predictor i	`FE_RE`, `FE_BR`
s_t	predetermined state vector of the original model	`meta.re_states` (IRIS `xbVector`)
k_x, c_x	coefficients of the anchored RE forecast	parameters `are_x__*`
$\mathbf{T}, \mathbf{K}, \mathbf{R}, \mathbf{Z}, \mathbf{H}, \mathbf{D}$	state-space matrices of a solved model	`sspace(m, 'triangular', false)`
ε_t	transition and measurement shocks	IRIS `eList`

The solved first-order form of any IRIS model used below is

$$\begin{bmatrix} x_t^f \\ x_t^b \end{bmatrix} = \mathbf{T}\, x_{t-1}^b + \mathbf{K} + \mathbf{R}\, \varepsilon_t, \qquad y_t = \mathbf{Z}\, x_t^b + \mathbf{H}\, \varepsilon_t + \mathbf{D}, \tag{1}$$

with x^f the non-predetermined and x^b the predetermined transition variables (including IRIS auxiliary lags such as `y{-1}`) and y the measurement variables. This is the form `sspace(m, 'triangular', false)` returns; the fast one-period step (Section 14.5), the anchored coefficients (Section 6.2), the time-varying decomposition (Section 17) and the Kalman history (Section 13.4) all read it.

2.2 The sequence of one quarter

Every stage that simulates the model (forecast, inversion history, density draws, evaluation runs, the Ramsey loss continuation) runs the same sequence in quarter t (`+engine/forecast.m`, `+engine/history.m`):

1. *Parameters.* The belief parameters are set from ϕ_{t-1} (`+plm/param_values.m`); the share parameters from `+engine/shares_for_mode.m`: $n^m_{1,t}$ and $n^m_{2,t} = 1 - n^m_{1,t}$. Under switching $n^m_{1,t} = n_{1,t-1}$, the share computed at the end of the previous quarter (Özden, 2025, eq. 16).
2. *Solve.* If the parameters differ from those of the last solve, the model is assigned and solved through the projection facility (`+engine/facility.m`, call `'solve'`). Normally the check of the actual law of motion (ALM, the model with the beliefs and shares in force; Section 12) at the end of $t-1$ has already solved this model, and its solution is reused.
3. *Simulate.* A one-period simulation of quarter t only, from the databank holding everything up to $t-1$, with the shocks of quarter t and the conditions active in t (`+engine/step.m`).
4. *Read the forecasts.* $f^{\mathrm{RE}}_t(x)$ and $f^{\mathrm{BR}}_t(x)$ are read from the simulated values of `x_exp1_RE` and `x_exp1_BR`. The learners' value is checked against the PLM formula with ϕ_{t-1} (tolerance 10^{-9}, error `hetlearn:engine:timing`).
5. *Update at the end of t* (`+engine/update_learning.m`): forecast errors FE^i_t of the forecasts made in $t-1$; fitness and shares $n_{1,t}$; RLS candidates for ϕ_t; the PLM part of the projection facility; the ALM check of the model in force in $t+1$; the update of the gain state.

Two assumptions hold throughout:

- *Anticipated utility* (Kreps, 1998). When the model is solved in quarter t, the beliefs ϕ_{t-1} and shares n^m_t are treated as permanent. Agents do not anticipate that beliefs or shares will change. The model is re-solved when they do.
- *Unanticipated shocks and conditions.* Each quarter is a new one-period simulation, so every shock and every condition is a surprise in the quarter it hits.

2.3 Information sets

Two options date the information behind the two forecasts made in t:

	`'lagged'`	`'current'`
`beliefs.info` (learners)	data up to $t-1$ (default)	data up to t (Özden, 2025, eq. 12)
`expectations.re_info` (RE agents)	$E_{t-1}x_{t+1}$	$E_t x_{t+1}$ (default)

Under both settings of `beliefs.info` the beliefs used in t are ϕ_{t-1} and the RLS update at the end of t uses regressors dated $t-1$ or earlier. Only the data that enter the forecast change. Sections 7.1 and 7.2 give the formulas.

3 Model requirements and the model package

3.1 What a model must have

- *An IRIS model file* that `Model.fromFile` loads. IRIS 20230622 is the tested release.
- *At least one forward-looking transition variable*: a declared transition variable with a lead in a transition equation or a substitution. Without one, `hetlearn.extend_model` stops with `hetlearn:extend_model:noLeads`.

- *A first-order solution.* The engine solves the model every time the beliefs or shares change and needs IRIS's state-space form (1). Linear models are the tested case (`spec.linear = true`). For a nonlinear model the code calls `spec.sstate` (or `sstate(m)`) before each `solve` (`+engine/solve_model.m`), so the path exists, but no package uses it and no test covers it. Two parts are linear-only: the anchored RE variant is refused for nonlinear models (`hetlearn:load_model:anchored`), and the backward simulator (Section 6.3) declines a nonlinear model. Since anchored RE is the default, a nonlinear model needs `expectations.re_variant = 'model_consistent'`.
- *Plain state names.* The anchored RE variant, the fast step and the Kalman history read IRIS's transition vector element by element and accept only `v` and `v{-k}`. Log variables (printed as `log(v)`) disable the anchored block (`meta.anchor_ok = false`) and stop the fast step and the Kalman history with an error.
- *At least as many shocks as observables* for the RE filter's fit check to apply, and for `history.mode = 'inversion'`.

The model's own equations are not restricted otherwise: hybrid terms, lags of any order, substitutions, measurement equations, the model's own `!if` and `!switch` blocks, `!for`, `!import` and `!include` are all accepted (Section 4.1).

3.2 The package contract: `model_spec.m`

A package is a folder `models/<name>/` (or any folder given by path) with a function `model_spec()` that returns a struct. `hetlearn.load_spec` evaluates it with the folder as the current folder, adds `spec.folder` and fills the optional fields (`load_spec.m`).

Field	Required	Meaning
`name`	yes	package name, used in generated file names
`model_file`	yes	IRIS model file, relative to the folder
`params`	yes	name of a function in the folder returning a struct of every parameter, `std_<shock>` and `corr_*` included
`linear`	yes	passed to `Model.fromFile`
`sstate`	no (`''`)	function name or handle `m = f(m)` called before every solve; `''` for linear models
`assign`	no (`struct()`)	values of the model file's own `!if`/`!switch` switches
`database`	for runs	CSV written by `databank.toCSV` or a `.mat` file (one struct, or the series as variables)
`filter_range`	for runs	`[first, last]` quarter of the RE filter and smoother
`observables`	for runs	cellstr of measurement variables
`policy`	no	Ramsey policy section (Section 15)
`report`	no	`labels`, `units`, `scale` per observable; `scale` converts model units to report units

Parameters are assigned by name: fields of the parameter struct that are not parameters, shock standard deviations (`std_<shock>`) or correlations (`corr_*`) of the model are ignored (`load_model.m`).

3.3 Package options

3.3.1 `model`

Default `''` (must be set; `hetlearn:run:model` otherwise).

Values a folder name under `models/`, or the path of any folder that holds a `model_spec.m` (absolute or relative to the current folder).

Math none.

Assumptions the package folder is self-contained; generated files go to `<folder>/generated/`.

Interactions `hetlearn.evaluate` requires every variant to use the same package (`hetlearn:evaluate:model`).

Code Example `load_spec.m`, `run.m`.

```
cfg.model = 'sw07';                      % models/sw07/model_spec.m
cfg.model = fullfile('tests', 'toy_nk');  % any folder with a model_spec.
    m
```

3.3.2 `model_params`

Default `''`: the package's `spec.params`. (Project baseline: `'sw07_params_original'`.)

Values `''`, or a valid MATLAB name of a parameter function that is a `.m` file in the package folder. Another name stops the load with `hetlearn:load_model:params`.

Math the parameter vector θ of the original model.

Assumptions parameters are given, never estimated by the toolbox.

Interactions the chosen function replaces `spec.params` everywhere downstream: the generator's collapse test (when it runs), the anchored coefficients, the policy discount, the RE filter (whose cache key hashes the parameter values, so each set has its own caches), and `hetlearn.evaluate`, which keeps parameter sets apart when it shares filters.

Code Example `load_model.m`, `re_filter.m`, `evaluate.m`.

```
cfg.model_params = 'sw07_params_original';   % published SW07 mode
cfg.model_params = '';                        % back to spec.params
```

4 The extended model

`hetlearn.extend_model(spec, plm)` reads the package's model file and writes an extended file in `<folder>/generated/`. `hetlearn.load_model` calls it before every run; it regenerates only when something changed (Section 4.5). The user never edits the generated file.

4.1 Lead detection

Code: `extend_model.m` (functions `mask_code`, `parse_structure`, `find_occurrences`, `active_leads`, `check_pseudofunctions`).

- Leads are detected only on *declared transition variables*, only inside `!transition_equations` (alias `!equations`) and `!substitutions`, and only in the dynamic part of an equation (the text before `!!`). A lead in a measurement equation is normalized to IRIS notation but not replaced.
- Comments (`%`, `%{ %}`, the rest of a line after `...`) and quoted labels are never changed. IRIS 20230622 accepts only `%` and `%{ %}`, so the delimiters `//`, `/*`, `*/` become `%%`, `%{`, `%}` with the comment text unchanged.
- Accepted lead notations, whitespace tolerant, for an integer $k \geq 1$: `x{+k}`, `x{k}`, `x{t+k}`, `x[+k]`, `x(+k)`. Non-IRIS lags (`x[-k]`, `x{t-k}`, `x(-k)`) are normalized to `x{-k}`. Because IRIS cannot load the non-IRIS forms, a reference file `<name>_norm.model` with only the subscripts normalized is written and used by the collapse test.
- The pseudo-functions `diff`, `dot`, `roc`, `pct`, `difflog`, `movsum`, `movprod`, `movavg`, `movgeom` with a lead inside or a forward window stop the generator (`hetlearn:extend_model:pseudoLead`), naming the equation. Rewrite them with explicit leads.
- Files with `!for`, `!import` or `!include` are expanded first with the IRIS preparser (`parser.Preparser.parse` with `spec.assign`); comments are lost and the model's own `!if` blocks are resolved in that case.
- A file with its own `!if`/`!switch` blocks keeps every branch, but the set $\mathcal{F}$ of forward-looking variables and their maximum leads are detected on the branches that `spec.assign` selects (the preparser resolves a copy of the file). A variable whose leads sit only in inactive branches is not forward-looking and keeps `x{+k}` there.
- Generated names that collide with declared names or with `spec.assign` fields stop the generator (`hetlearn:extend_model:nameCollision`).

For each $x \in \mathcal{F}$ the generator records the maximum lead K_x (`meta.max_lead`). $\mathcal{F}$ is listed in declaration order (`meta.forward_vars`).

4.2 Rewriting and the expectation block

Every lead `x{+k}` with $x \in \mathcal{F}$ becomes `x_expk`. The generator then appends an expectation block (new declaration and equation sections at the end of the file; IRIS allows repeated sections, so the user's sections are not touched). For the default AR(1) PLM the block of one variable reads (`make_block` in `extend_model.m`):

```
!transition_variables
    "Aggregate expectation of x{+1}" x_exp1
    "Aggregate expectation of x{+2}" x_exp2              % only if K_x >= 2
    !if br_x
        "RE expectation of x{+1}" x_exp1_RE
        "BR (adaptive learning) expectation of x{+1}" x_exp1_BR
    !end
!parameters
    n1_x, n2_x, alpha_x, beta_x, are_x__const, are_x__<state>, ...
!transition_equations
```

```
!if br_x
    x_exp1 = n1_x*x_exp1_RE + n2_x*x_exp1_BR;
    !if re_anchored
        x_exp1_RE = are_x__const + are_x__<s1>*s1 + ...;
    !else
        x_exp1_RE = x{+1};
    !end
    !if br_info_current
        x_exp1_BR = alpha_x + beta_x*x;
    !else
        x_exp1_BR = alpha_x + beta_x*(alpha_x + beta_x*x{-1});
    !end
!else
    x_exp1 = x{+1};
!end
x_exp2 = x_exp1{+1};                                    % only if K_x >= 2
```

In equations, for $x \in \mathcal{L}$ (`br_x = true`),

$$\tilde{E}_t x_{t+1} = n^m_{1,t}\, f^{\mathrm{RE}}_t(x) + n^m_{2,t}\, f^{\mathrm{BR}}_t(x), \qquad n^m_{2,t} = 1 - n^m_{1,t}, \tag{2}$$

$$\tilde{E}_t x_{t+k} = E_t\big[\tilde{E}_{t+1} x_{t+k}\big], \qquad k = 2, \ldots, K_x, \tag{3}$$

and for $x \in \mathcal{F} \setminus \mathcal{L}$ (`br_x = false`) the expectation is the model-consistent one, $\tilde{E}_t x_{t+1} = E_t x_{t+1}$, as in the original model. The engine always sets $n_2 = 1 - n_1$ (`+engine/facility.m`, call `'params'`).

Equation (3) is the IRIS lead of an endogenous variable: the k-step aggregate expectation is the model-consistent forecast of next quarter's $(k-1)$-step aggregate expectation. Heterogeneity therefore enters at the one-step level, and longer expectations are built by iterating it. A model with a lead of two or more keeps a lead even when every forward-looking variable learns.

Derived regressors. Each `beliefs.plm.derived.<name>` adds the transition variable `<name>` with the equation `<name> = <expression>;`, outside every toggle (Section 8.2.3).

Log variables. If x is a log variable (`!log_variables`), the aggregate expectations `x_exp1`, ..., `x_expK` are declared as log variables too. With `!all_but` a warning (`hetlearn:extend_model:logAllBut`) says that they inherit the log status.

The lagged RE file. With `expectations.re_info = 'lagged'` the generator writes a second file whose RE expectation uses information up to $t-1$ (Section 7.2).

4.3 Model-file toggles

The block sits behind IRIS `!if` switches. `hetlearn.load_model` sets them from the options through the `'assign'` argument of `Model.fromFile`:

Switch	Set from	Effect when true
`br_<x>` (one per $x \in \mathcal{F}$)	`expectations.variables`	x's expectation is the RE/learner mix (2); false: $E_t x_{t+1}$
`re_anchored` (global)	`expectations.re_variant`	anchored RE forecast (Section 6.2) instead of `x{+1}`
`br_info_current` (global)	`beliefs.info`	learners' forecast on data up to t
`ramsey` (global)	`policy.mode`	the Ramsey loss replaces the policy rule (Section 15)

The model's own switches come from `spec.assign`. `expectations.re_info` is not a switch: it selects another generated file.

4.4 Parameters of the extended model

Every generated parameter is declared unconditionally (an unused parameter is harmless):

- shares `n1_x`, `n2_x` for every $x \in \mathcal{F}$;
- the belief parameters of every PLM equation of the generator's PLM system (Section 8.1). The generator does not know the scenario's learning set, so it gives *every* forward-looking variable a primary PLM, built from `beliefs.plm`, plus the auxiliary PLMs that close the system;
- the anchored coefficients `are_x__const`, `are_x__<v>` and `are_x__<v>_l<k>`, one per state of the original model (Section 6.2); left out when `meta.anchor_ok` is false;
- with `spec.policy`: the loss weights `w_<k>`, `ramsey_disc_on` and `ramsey_disc` (Section 15).

At load time every `n1_x` is 1, every `n2_x` 0, every belief parameter 0, and the anchored coefficients take their computed values (`load_model.m`). The loaded model is then the RE model (Section 4.6); the engine assigns beliefs and shares each quarter.

4.5 Generated files and the cache

File (in `generated/`)	Content
`<name>_ext_<tag>.model`	the extended model (generator version 4)
`<name>_ext_<tag>_meta.mat`	`meta`: source file and MD5, normalized PLM spec and key, $\mathcal{F}$, maximum leads, generated names, toggles, PLM system, states of the original model, anchored parameter names, policy section, collapse-test results, generator version
`<name>_ext_<tag>_relag.model`, `_relag_meta.mat`	the lagged RE file (version 5)
`<name>_norm.model`	the source with subscripts in IRIS notation (only when needed)

The tag is `ar1` when every PLM equation is an AR(1) with constant on the own lag and there are no derived regressors; otherwise `p` followed by the first 8 hexadecimal digits of the MD5 of the normalized PLM spec. A cached file is reused only if the generator version, the source MD5, the PLM key, the `spec.assign` key and the policy key all match; otherwise it is rewritten.

4.6 The collapse test

At $n_1 = 1$, $n_2 = 0$ and zero beliefs, equation (2) gives $\tilde{E}_t x_{t+1} = f_t^{\mathrm{RE}}(x)$. With `re_anchored = false` this is `x{+1}`, so the extended model is the original model with renamed leads. With `re_anchored = true` and the anchored coefficients of the original model, it is the RE forecast at the current state, which on the RE path is again $E_t x_{t+1}$. Every generated file must pass (`collapse_test` in `extend_model.m`):

1. every `br_x = true`, `re_anchored = false`;
2. every `br_x = false`;
3. every `br_x = true`, `re_anchored = true`, coefficients from `hetlearn.re_anchor_coefficients` (only when the file has the anchored block);
4. every `br_x = true`, `br_info_current = true`.

Each pass compares the impulse responses of every variable of the reference model to every shock over 40 quarters; the largest absolute difference must be below 10^{-10} (`hetlearn:extend_model:collapseFailed` otherwise). The results are stored in `meta.collapse_maxdiff` (passes 1–3) and `meta.collapse_info_maxdiff` (pass 4). The lagged file has its own passes (Section 7.2).

4.7 Generator call options

`meta = hetlearn.extend_model(spec, plm, ...)` takes the call options `'force'` (default false), which regenerates the file even when the cache matches, and `'validate'` (default true), which runs the collapse test. `'re_info'` selects the lagged file and is set by `hetlearn.load_model`. A direct call with a vector autoregression (VAR) over `'learning'` gives every forward-looking variable the VAR, because the learning set is resolved only by `hetlearn.load_model` (Section 8.2.4).

5 Expectations: the aggregate and the modes

Two options decide which expectations are heterogeneous and how the population is split: the learning set $\mathcal{L}$, and the mode, which fixes the RE share $n_{1,t}^m$ in the aggregate expectation (2).

5.1 Options

5.1.1 `expectations.mode`

Default `'rational'`.

Values `'rational'`, `'learning'`, `'fixed_shares'`, `'switching'`.

Math the RE share in the model in quarter t (`+engine/shares_for_mode.m`):

$$n_{1,t}^m = \begin{cases} 1 & \texttt{'rational'} \\ 0 & \texttt{'learning'} \\ 1 - \bar{n}_2 & \texttt{'fixed_shares'},\ \bar{n}_2 = \texttt{shares.fixed_n2} \\ n_{1,t-1} & \texttt{'switching'}, \text{ the share of the variable's group} \end{cases} \tag{4}$$

In every mode the end-of-quarter update still computes the fitness and the discrete-choice shares $n_{1,t}$ (Section 11) and, under `beliefs.update = 'rls'`, the beliefs; outside `'switching'` they are diagnostics that do not enter the model.

Assumptions `'rational'`: every agent is RE, so the run is exactly the RE model of the scenario (the collapse argument of Section 4.6); with the default unanticipated shocks it equals one IRIS `simulate` of the RE model over the horizon (test `test_engine` E1). `'switching'`: shares follow the discrete-choice rule; the share in force in t was computed at the end of $t-1$.

Interactions `'fixed_shares'` needs `shares.fixed_n2` (`hetlearn:options:missing`). The projection facility skips its ALM check when every learners' model share is 0 (`'rational'`, or fixed shares of 0). The density fast path needs `'rational'` (Section 16); `hetlearn.decompose` with `method = 'auto'` uses IRIS contributions for `'rational'` (Section 17). With an empty learning set the mode has no effect. A share kept by the facility (Section 12.3) overrides (4) for one quarter.

Code Example `+engine/shares_for_mode.m`, `+engine/update_learning.m`.

```
cfg.expectations.mode = 'switching';
cfg.expectations.variables = {'pinf'};
```

5.1.2 `expectations.variables`

Default `{}` (no learning variable: the run is RE whatever the mode).

Values a cellstr of forward-looking transition variables, or `'all'` for every $x \in \mathcal{F}$. A char is turned into a one-element cellstr.

Math the learning set $\mathcal{L}$; `br_x = true` for $x \in \mathcal{L}$.

Assumptions each $x \in \mathcal{L}$ gets its own learners' forecast and its own PLM; variables outside $\mathcal{L}$ keep $E_t x_{t+1}$, the model-consistent expectation of the heterogeneous economy (the anchored variant applies only inside the `br_x` blocks).

Interactions a name that is not forward-looking stops the load (`hetlearn:load_model:notForward`; `info.meta.forward_vars` lists the candidates). $\mathcal{L}$ is ordered as declared in the model, not as listed. It defines the share groups (`shares.structure`), the default of `gain.koz.master`, the members of a VAR over `'learning'`, and, with `expectations.re_info = 'lagged'`, the RE(t-1) model of Section 7.2 (an empty set is then an error). With anchored RE and every forward-looking variable of maximum lead 1 in $\mathcal{L}$, the model has no leads (Section 6.3).

Code Example `load_model.m` (`learning_vars`), `extend_model.m`.

```
cfg.expectations.variables = {'pinf', 'w'};
cfg.expectations.variables = 'all';          % every forward-looking
    variable
```

6 RE agents

6.1 Model-consistent RE agents

With `re_anchored = false` the generated equation is `x_exp1_RE = x{+1}`:

$$f_t^{\mathrm{RE}}(x) = E_t\left[x_{t+1} \,\middle|\, \phi_{t-1},\, n_t^m\right], \tag{5}$$

the rational forecast in the heterogeneous economy whose ALM IRIS solves in quarter t, under the anticipated-utility assumption of Section 2.2. RE agents know the model, the learners' PLM coefficients and the shares.

6.2 Anchored RE agents

Following Özden (2025, sec. 2.3.1), anchored RE agents use the forecast of the pure-RE model, evaluated at the current state of the heterogeneous economy. Write the RE solution of the original model at its parameters as (1) and let $s_t = x_t^b$ be its predetermined state vector (IRIS `xbVector`: current values of variables that appear with lags, auxiliary lags such as `c{-1}`, the exogenous processes). Then $E_t^{\mathrm{RE}} x_{t+1} = \mathbf{K}_x + \mathbf{T}_{x\cdot} s_t$, and

$$f_t^{\mathrm{RE}}(x) = k_x + c_x' s_t, \qquad k_x = \mathbf{K}_x, \quad c_x = \mathbf{T}_{x\cdot}', \tag{6}$$

with s_t taken from the heterogeneous economy. The generated equation is `x_exp1_RE = are_x__const + sum_j are_x__<s_j>*s_j`, where a state `v` gets the parameter `are_x__v` and an auxiliary lag `v{-k}` the parameter `are_x__v_lk` (`+plm/anchor_names.m`).

The coefficients come from `hetlearn.re_anchor_coefficients` (`re_anchor_coefficients.m`, through `anchor_coefficients.m` with lag 0 and horizon 1) on the original model at `spec.params` (or `model_params`). They are checked by unit-perturbation simulation: one IRIS `simulate` of quarter $t+1$ with zero shocks from a databank with $n_b + 1$ columns, the state at 0 and at each unit vector, must return k_x and $k_x + c_{x,j}$ (relative tolerance 10^{-9}; `hetlearn:anchor:check`). `hetlearn.load_model` recomputes them at every load, so they follow the parameters.

Assumptions. Anchored agents believe the economy is the pure-RE economy and keep its reduced form; they do not internalize the learners or the shares. At $n_1 = 1$ the anchored model is the RE model (collapse pass 3). The forecast (6) uses s_t, which holds values dated t, so it is determined jointly with the other variables dated t. Under Ramsey policy the coefficients come from the rule-based RE model (Section 15).

6.3 Existence and models without leads

When every forward-looking variable is in $\mathcal{L}$, every maximum lead is 1 and the RE agents are anchored, the ALM has no leads (`get(m, 'maxLead') = 0`). A linear model without leads has a unique solution for any beliefs and shares whenever its system in the variables dated t is nonsingular, possibly explosive. IRIS rejects explosive solutions even in purely backward models, so when IRIS fails on such a model, `+engine/solve_model.m` builds a backward system (`+engine/backward_simulate.m`) from `system(m)`:

$$A \begin{bmatrix} x_{t+1}^f \\ x_t^b \end{bmatrix} + B \begin{bmatrix} x_t^f \\ x_{t-1}^b \end{bmatrix} + C + D\varepsilon_t = 0, \qquad F y_t + G x_t^b + H + J\varepsilon_t = 0, \tag{7}$$

$$M = [B_{\cdot,1:n_f},\ A_{\cdot,n_f+1:n}], \qquad \begin{bmatrix} x_t^f \\ x_t^b \end{bmatrix} = -M^{-1}(B_b x_{t-1}^b + C + D\varepsilon_t), \tag{8}$$

$$y_t = -F^{-1}(G x_t^b + H + J\varepsilon_t), \tag{9}$$

with $B_b = B_{\cdot,n_f+1:n}$ and no leads meaning $A_{\cdot,1:n_f} = 0$. The roots are the eigenvalues of Φ, the rows of $-M^{-1}B_b$ that belong to x^b. What happens next depends on the projection facility (Section 12): under `'none'` the quarter is simulated with this system and an `explosive` event records the largest root; under `'alm'` and `'plm_alm'` a backward system with a root above 1 in modulus is an unstable ALM. The backward system equals IRIS `simulate` whenever IRIS has a solution (`test_anchored` A3). A model with a lead of two or more keeps `x_exp2 = x_exp1{+1}` and so keeps a lead.

6.4 Option

6.4.1 `expectations.re_variant`

Default `'anchored'`.

Values `'anchored'`, `'model_consistent'`.

Math `'anchored'`: equation (6); with `re_info = 'lagged'`, equation (13). `'model_consistent'`: equation (5); with `'lagged'`, the chain (11).

Assumptions see Sections 6.1–6.2. Under the Taylor rule the two variants give the same RE path (both are the RE model at $n_1 = 1$).

Interactions the variant applies to the learning set only. `'anchored'` needs a linear model with plain state names (`hetlearn:load_model:anchored`); `'model_consistent'` works for any model IRIS solves. With anchored RE and every lead-1 forward-looking variable learning, the model has no leads (Section 6.3). For an RE Ramsey benchmark see Section 15.5.1. With many learning variables a model-consistent ALM can lose its stable solution: on SW07 with all twelve forward-looking variables learning, such a run completes only because the facility keeps the model shares (look for `projection_alm_shares` in `events.csv`). The RE filter of the history uses the scenario's variant (its own cache file).

Code `extend_model.m` (`anchor_states`, `anchor_expr`), `re_anchor_coefficients.m`, `anchor_coefficients.m`, `load_model.m`.

Example

```
cfg.expectations.re_variant = 'model_consistent';
cfg.policy.mode = 'ramsey';                    % RE Ramsey benchmark
```

7 Information timing

7.1 The learners' information set

The learners' forecast of x_{t+1} is built by iterating the PLMs forward (`+plm/forecast_expr.m` writes it as text for the model file, `+plm/forecast_value.m` evaluates it numerically). With the PLM equation of variable v, $v_t = \alpha_v + \sum_j \beta_{v,j}\, u_{j,t-l_j}$ (Section 8), define

$$\mathcal{E}(v,h) = \alpha_v + \sum_j \beta_{v,j}\, \mathcal{T}(u_j,\, h - l_j), \qquad \mathcal{T}(u,k) = \begin{cases} u_{t+k} & k \le k_{\max} \\ \mathcal{E}(u,k) & k > k_{\max} \end{cases} \tag{10}$$

and $f_t^{\mathrm{BR}}(x) = \mathcal{E}(x,1)$, all with the coefficients ϕ_{t-1}. The information set fixes $k_{\max}$: -1 under `'lagged'` (data up to $t-1$), 0 under `'current'` (data up to t). Since every regressor lag is at least 1, under `'current'` $\mathcal{E}(x,1)$ never recurses. Examples:

PLM	`'lagged'`	`'current'`
AR(1)	$\alpha + \beta(\alpha + \beta x_{t-1})$	$\alpha + \beta x_t$
AR(2)	$\alpha + \beta_1(\alpha + \beta_1 x_{t-1} + \beta_2 x_{t-2}) + \beta_2 x_{t-1}$	$\alpha + \beta_1 x_t + \beta_2 x_{t-1}$
VAR(1)	$a + B(a + BX_{t-1})$	$a + BX_t$

Under `'current'` the forecast contains x_t and is solved jointly with it (Özden, 2025, eq. 12). The beliefs used in t are ϕ_{t-1} under both settings, and the RLS update at the end of t uses

the same regressors (dated $t-1$ or earlier), so the beliefs do not depend on the information set; the forecasts, and through them the fitness and the shares, do. Under `'current'` the error that enters fitness at t compares x_t with $f^{\mathrm{BR}}_{t-1}(x) = \alpha_{t-2} + \beta_{t-2}x_{t-1}$ (AR(1)).

7.1.1 `beliefs.info`

Default `'lagged'`.

Values `'lagged'`, `'current'`.

Math $k_{\max} = -1$ or 0 in (10).

Assumptions `'lagged'`: learners observe data up to $t-1$ when they forecast in t. `'current'`: they observe x_t and the other regressors dated t, as in Özden (2025).

Interactions the generated file holds both forms; `br_info_current` selects one (a file from generator version 3 or older with `'current'` is `hetlearn:load_model:info`). Under `gain.koz.error = 'forecast'` the error that drives the gain is $x_t - f^{\mathrm{BR}}_{t-1}(x)$, a two-step error under `'lagged'`. In the history stage the forecast reads the history databank at $t+k$ with $k \le k_{\max}$; in the Kalman history it reads the real-time filtered values.

Code Example `+plm/forecast_expr.m`, `+plm/forecast_value.m`, `load_model.m`.

```
cfg.beliefs.info = 'current';                % alpha + beta*x_t
```

7.2 The RE agents' information set

With `expectations.re_info = 'lagged'` the RE expectation of every $x \in \mathcal{L}$ is $E_{t-1}x_{t+1}$, formed with information up to $t-1$; expectations of other variables stay E_t. This is the RE counterpart of `beliefs.info = 'lagged'`: both forecasts made in t then use data up to $t-1$, and both fitness errors are two-step errors. Every RE object of a lagged scenario refers to the *RE(t-1) model of its learning set*: the original model with $E_{t-1}x_{t+1}$ in place of E_tx_{t+1} for $x \in \mathcal{L}$, solved as an RE fixed point. That model is the RE benchmark (mode `'rational'`), the anchored agents' model, the model of the RE filter and smoother, and the source of the RE-moment initial beliefs.

Model-consistent form. The lagged file (`make_block_lagged` in `extend_model.m`) writes, inside each `br_x` block,

$$x^{(1)}_t = x_{t+1}, \qquad x^{(2)}_t = x^{(1)}_{t+1}, \qquad f^{\mathrm{RE}}_t(x) = x^{(2)}_{t-1}, \tag{11}$$

(`x_REt1 = x{+1}`, `x_REt2 = x_REt1{+1}`, `x_exp1_RE = x_REt2{-1}`), so $x^{(1)}_t = E_tx_{t+1}$, $x^{(2)}_t = E_tx_{t+2}$ and $f^{\mathrm{RE}}_t(x) = E_{t-1}x_{t+1}$. Every lead stays at +1 (IRIS's own `x{+2}` would add a lead element the fast step cannot read). With model-consistent agents, $x^{(2)}_{t-1}$ comes from the model in force at $t-1$, so $f^{\mathrm{RE}}_t(x)$ is predetermined in t.

Anchored form. Write the solution of the model-consistent lagged model at $n_1 = 1$ and zero beliefs as (1), with $x^b = (s, z, o)$: s the original model's states, z the chain states $z_{x,t} = x^{(2)}_t$, o any other state (PLM lags, derived regressors). Iterating two quarters from the state at $t-1$,

$$E_{t-1}x_{t+1} = \mathbf{K}_x + \mathbf{T}_{x\cdot}(\mathbf{K}_b + \mathbf{T}_b x^b_{t-1}) = k + C_s s_{t-1} + C_z z_{t-1}, \tag{12}$$

where the loadings on o must be below 10^{-12} (`hetlearn:anchor:lagged`). In the heterogeneous economy the chain state z_{t-1}, the RE agents' own forecast of x_{t+1} formed with information up to $t-1$, is f_t^{RE} itself. Stacking the learning set, $f_t = k + C_s s_{t-1} + C_z f_t$, so

$$f_t^{\mathrm{RE}} = (I - C_z)^{-1}(k + C_s s_{t-1}), \tag{13}$$

a linear function of states dated $t-1$ and earlier (`re_anchor_coefficients.m`, `lagged_coefficients`; a near-singular $I - C_z$, reciprocal condition below 10^{-12}, is an error). The generated equation dates every state one quarter earlier: `x_exp1_RE = are_x__const + sum_j are_x__<s_j>*s_j{-1}`, a lag-k state becoming `v{-k-1}`. On the RE(t-1) path z_{t-1} satisfies the same fixed point, so at $n_1 = 1$ the anchored and model-consistent lagged models coincide; `hetlearn.load_model` checks it at every load (impulse responses of every common variable, 40 quarters, below 10^{-10}, `info.re_info_check`). The coefficients depend on the learning set, so the model-consistent lagged model is loaded and solved for every anchored lagged scenario.

Collapse test of the lagged file. (`collapse_test_lagged`.) (1) every `br_x = false` reproduces the original model; (2) with every lead-1 forward-looking variable learning and `re_anchored = false`, the RE(t-1) model of that set must solve; (3) the same with `re_anchored = true` and the lagged coefficients reproduces (2); (4) (2) with `br_info_current = true` reproduces (2). If (2) has no unique stable solution for the whole set, passes (2)–(4) run one variable at a time. `meta.collapse_maxdiff(1)` is NaN, since the lagged file differs from the original by construction.

History. The RE filter of a lagged scenario keeps the scenario's learning set and variant (own cache), and also returns the one-step predictions $x_{t|t-1}$ (`H.predicted`). Under `history.re_states = 'smoothed'` the RE forecast of the history is the smoothed `x_exp1_RE` at t, which equals the two-step forecast from the smoothed state at $t-1$. Under `'filtered'` it is `H.predicted.x_exp1_RE(t)` $= E[x_{t+1} \mid \text{data to } t-1]$, the filter's two-step prediction (`+engine/history_db.m`); the learners' data and the realized values stay $x_{t|t}$.

7.2.1 `expectations.re_info`

Default `'current'`. (Project baseline: `'lagged'`.)

Values `'current'`, `'lagged'`.

Math `'current'`: $f_t^{\mathrm{RE}}(x)$ from (5) or (6). `'lagged'`: from (11) or (13).

Assumptions under `'lagged'` the RE agents see the same data as lagged learners; the RE benchmark becomes the RE(t-1) model of $\mathcal{L}$, not the original model.

Interactions `'lagged'` needs a non-empty learning set (`hetlearn:options:value`), learning variables with a maximum lead of 1 (`hetlearn:load_model:reInfo`), and `policy.mode = 'rule'` (`hetlearn:options:unsupported`). It changes the RE filter, the RE forecasts of the history (`H.predicted` under `'filtered'`), the anchored coefficients, the RE-moment initial beliefs and the Kalman history (RE forecasts from the predictions $x_{t|t-1}$). `hetlearn.evaluate` gives a lagged variant its own real-time filters (the group key adds `re_info`, the learning set and `re_variant`). Compare a lagged learning run with a lagged RE run.

Code `extend_model.m` (`make_block_lagged`, `collapse_test_lagged`), `re_anchor_coefficients.m`, `load_model.m`, `re_filter.m`, `+engine/history_db.m`.

Example

```
cfg.expectations.re_info = 'lagged';        % E_{t-1} x_{t+1} for the
    learning set
cfg.beliefs.info = 'lagged';                % learners on the same data
```

8 Perceived laws of motion

8.1 The PLM equation

Each learning variable x has a primary PLM, and every regressor without its own PLM gets an auxiliary one, so that the system is closed (`+plm/build_system.m`). A PLM equation for a target v is

$$v_t = \alpha_v + \sum_{j=1}^{J_v} \beta_{v,j}\, u_{j,t-l_j} + e_{v,t}, \qquad l_j \geq 1, \tag{14}$$

with regressor vector and coefficients

$$z_{v,t-1} = (1,\, u_{1,t-l_1},\, \ldots,\, u_{J,t-l_J})', \qquad \phi_v = (\alpha_v,\, \beta_{v,1},\, \ldots,\, \beta_{v,J})'. \tag{15}$$

Without a constant the leading 1 and α_v are dropped. Every regressor is dated $t-1$ or earlier, whatever the information set; `beliefs.info` only changes which data enter the forecast (10). Regressors are lags of transition variables of the extended model, including derived regressors (Section 8.2.3).

Closing the system. Starting from the learning variables, the builder walks a queue: a VAR member gets the VAR equation; a variable with `beliefs.plm.by_var.<v>` gets that specification; a learning variable otherwise gets `beliefs.plm.default`; any other regressor gets an AR(1) with constant. Each new regressor joins the queue until every regressor has an equation. Equations added this way are *auxiliary* (`isAux = true`); they are learned by the same RLS and recorded like the primary ones, but no agent uses their forecast except inside the recursion (10).

Parameter names. An AR(1) with constant on the own first lag has `alpha_x` and `beta_x`. Any other PLM has `alpha_x` (if it has a constant) and one `beta_x__<u>_l<l>` per regressor, e.g. `beta_pinf__ygap_l1`. Coefficient vectors are ordered like the names: constant first, then the regressors in the order given (a VAR: lag first, then members in the order listed). The same names are the model parameters, the fields of `res.learning.<target>` and the rows of `learning.csv`.

Consistency of the generated file and the run. The generator builds its PLM system with every forward-looking variable as a primary target (Section 4.4); the run builds it with the scenario's learning set. `hetlearn.load_model` compares, for every $x \in \mathcal{L}$, the forecast text the run's system gives with the one in the active branch of the generated file, and checks that every belief parameter is declared (`hetlearn:load_model:plmMismatch` otherwise). The two differ when a learning variable's PLM uses a forward-looking variable $u \notin \mathcal{L}$ as a regressor and `beliefs.plm.default` is not `'ar1'`: the generator gives u the default PLM as a primary target, the run gives it an auxiliary AR(1). Setting `beliefs.plm.by_var.<u>` makes the two agree.

8.2 Options

8.2.1 `beliefs.plm.default`

Default `'ar1'`.

Values `'ar1'`, `'ar2'`, `'ar1_noconst'`, `'var1'`.

Math for a learning variable x without a `by_var` entry and outside a VAR: `'ar1'` $x_t = \alpha + \beta x_{t-1}$; `'ar2'` $x_t = \alpha + \beta_1 x_{t-1} + \beta_2 x_{t-2}$; `'ar1_noconst'` $x_t = \beta x_{t-1}$; `'var1'` one VAR(1) with constant over $\mathcal{L}$, $X_t = a + BX_{t-1}$ with $X = (x)_{x \in \mathcal{L}}$.

Assumptions learners fit a univariate (or VAR) reduced form; they do not know the model's structure.

Interactions auxiliary PLMs stay AR(1) with constant whatever the default (set them with `by_var`); see the consistency check above. `'var1'` together with `beliefs.plm.var.variables` is an error (`hetlearn:options:value`); `'var1'` ignores `beliefs.plm.var.lags` and `.constant` (it is always lag 1 with a constant) and is resolved to $\mathcal{L}$ before generation, so the generated file depends on the learning set. Any default other than `'ar1'` gives a new file tag.

Code `+plm/build_system.m` (`one_spec`, `var_spec`), `load_model.m` (`resolve_learning_set`).

Example

```
cfg.beliefs.plm.default = 'ar2';
cfg.beliefs.plm.default = 'var1';          % VAR(1) over the learning
    set
```

8.2.2 `beliefs.plm.by_var`

Default `struct()` (no entries).

Values `by_var.<v>` is a shorthand (`'ar1'`, `'ar2'`, `'ar1_noconst'`) or a struct with fields `regressors` (cellstr of `'u{-l}'`, $l \geq 1$; `u(-l)`, `u[-l]` and `u{t-l}` are read too) and `constant` (logical, default true). `<v>` is any transition variable or derived name except a VAR member.

Math equation (14) with the listed regressors. With `regressors = {}` and a constant the PLM is mean-only, $v_t = \alpha_v + e_{v,t}$, and the forecast is $\alpha_{v,t-1}$. With $z = 1$ and $R_0 = 1$ (the RE-moment value) RLS keeps $R_t = 1$ and becomes exponential smoothing, $\alpha_t = \alpha_{t-1} + g_t(v_t - \alpha_{t-1})$: adaptive expectations.

Assumptions the learners of v regress on the listed lags only.

Interactions an entry for a learning variable sets its primary PLM; an entry for any other variable sets its auxiliary PLM. Errors: a VAR member (`hetlearn:plm:varByVar`), an unknown variable (`hetlearn:plm:unknownVar`), a regressor that is not a transition variable or has a lag below 1 (`hetlearn:plm:regressor`), duplicate regressors, neither constant nor regressors (`hetlearn:plm:empty`), unknown fields (`hetlearn:plm:field`). The subtree is free-form (merged without name checks, Section 1.5); the builder checks the names against the model. A regressor that is a forward-looking variable outside $\mathcal{L}$ may need its own entry (consistency check above).

Code `+plm/build_system.m`.

Example

```
cfg.beliefs.plm.by_var.pinf = struct('regressors', {{'pinf{-1}', 'ygap
    {-1}', 'r{-1}'}}, 'constant', true);
cfg.beliefs.plm.by_var.r = 'ar2';          % auxiliary PLM of the
    regressor r
```

8.2.3 `beliefs.plm.derived`

Default `struct()`.

Values `derived.<name> = '<expression>'`: a non-empty string of current-dated declared names, numbers, `+ - * /` and parentheses, with at least one transition variable; no time subscripts and no functions. `<name>` must be a new name.

Math a new transition variable $d_t = \text{expression}_t$, written as `<name> = <expression>;` outside every toggle, so it exists in the RE model and in every learning model.

Assumptions learners can observe the derived quantity (e.g. an output gap) with the same timing as other variables.

Interactions a derived variable is a regressor at any lag ≥ 1 or a VAR member; it gets an auxiliary AR(1) PLM like any other regressor. It is dated t in the forecast under `beliefs.info = 'current'`. It changes the model text, hence the file tag and the RE filter's cache key; its smoothed path is in the history. Errors: bad syntax or unknown names (`hetlearn:extend_model:derived`), a name collision (`hetlearn:extend_model:nameCollision`), a non-string value (`hetlearn:options:type`).

Code `extend_model.m` (`derived_defs`), `+plm/build_system.m`.

Example

```
cfg.beliefs.plm.derived.ygap = 'y - yf';
cfg.beliefs.plm.by_var.pinf = struct('regressors', {{'pinf{-1}', 'ygap
    {-1}'}}, 'constant', true);
```

8.2.4 `beliefs.plm.var`

Default `struct('variables', {{}}, 'lags', 1, 'constant', true)`: no VAR.

Values `variables`: a cellstr of transition or derived variables, or `'learning'` ($= \mathcal{L}$); `lags` p: a positive integer; `constant`: true or false.

Math for every member $v \in V$,

$$v_t = \alpha_v + \sum_{l=1}^{p} \sum_{u \in V} \beta_{v,u,l}\, u_{t-l} + e_{v,t}, \tag{16}$$

parameters `alpha_v` and `beta_v__u_l<l>`. Learning members use it for their forecast; the forecast iterates the VAR two steps under `'lagged'` information (10). Members outside $\mathcal{L}$ are auxiliary equations of the same VAR.

Assumptions learners use a joint reduced form. Each equation has its own RLS and moment matrix; with common regressors, a common gain and equal initial moment matrices, this equals system RLS.

Interactions `'learning'` is resolved to $\mathcal{L}$ by `hetlearn.load_model` before generation (a direct `hetlearn.extend_model` call would use every forward-looking variable). A `by_var` entry for a member is an error; a member listed twice or not in the model is an error; `default = 'var1'` with `variables` set is an error. The VAR is one block of the projection facility. RE-moment initial beliefs use every cross-moment. Under KOZ `'per_variable'` each equation has its own gain process.

Code `+plm/build_system.m` (`var_spec`, `var_eq`), `load_model.m`.

Example

```
cfg.beliefs.plm.derived.ygap = 'y - yf';
cfg.beliefs.plm.var = struct('variables', {{'pinf', 'ygap', 'r'}}, 'lags'
    , 1, 'constant', true);
```

9 Belief updating

9.1 Recursive least squares

After x_t is realized, each PLM equation v moves its beliefs by RLS (Özden, 2025, eq. 13), (Evans and Honkapohja, 2001):

$$R_{v,t} = R_{v,t-1} + g_{v,t}\left(z_{v,t-1}z'_{v,t-1} - R_{v,t-1}\right), \tag{17}$$

$$\phi_{v,t} = \phi_{v,t-1} + g_{v,t}\, R_{v,t}^{-1} z_{v,t-1}\left(v_t - \phi'_{v,t-1}z_{v,t-1}\right). \tag{18}$$

R is updated first and ϕ uses the new R_t (a linear solve, no inverse; `+rls/update.m`). Equation by equation, each with its own R. ϕ_t is in force in $t+1$. The residual $e_{v,t} = v_t - \phi'_{v,t-1}z_{v,t-1}$ is stored (`S.fe_hist`) and is the error that drives a KOZ gain under `gain.koz.error = 'rls'`.

Order inside the update (`+engine/update_learning.m`). The update runs step 5 of Section 2.2 in the order listed there. The gain of each equation, $g_{v,t}$, comes from the gain state before this quarter's update and is computed before the RLS candidates; the gain state itself is updated last, whatever the projection facility did.

Missing data. An equation whose target or any regressor is not finite at t keeps its beliefs and logs an `rls_missing` event. This happens only where the databank the stage reads holds a NaN; the smoothed and filtered RE histories, the inversion and the Kalman history fill quarters without data with model values.

The first quarter. At $t =$ `beliefs.init.date` there are no forecast errors (the forecasts of $t-1$ were not made), so fitness and shares keep their initial values, but the RLS update uses the data of that quarter.

9.1.1 `beliefs.update`

Default `'rls'`.

Values `'rls'`, `'fixed'`. (The removed values `'sac'` and `'kf'` raise `hetlearn:options:removed`.)

Math `'rls'`: (17)–(18). `'fixed'`: $\phi_t = \phi_0$, $R_t = R_0$ for all t.

Assumptions `'fixed'`: learners keep their initial rule; forecasts still move with the data and shares still switch.

Interactions with `'fixed'` no gain is computed, the gain state stays unchanged, and the PLM part of the facility has nothing to check; the ALM check can still keep the previous model shares (the belief revert is skipped because nothing changed). `'fixed'` with `beliefs.init.method = 'user'` gives a run with given forecasting rules.

Code `+engine/update_learning.m`.

Example

```
cfg.beliefs.update = 'fixed';
cfg.beliefs.init.values.pinf = struct('alpha', 0.5, 'beta', 0.9);
```

9.2 Gain rules

The gain $g_{v,t}$ of each equation comes from the function `hetlearn.gain.<rule>` named by `gain.rule`. Every rule answers four calls (`+gain/constant.m` documents the interface): `'fields'` (extra series it records), `'init'` (initial state), `'gain'` (the gains of this quarter's RLS updates, called before them) and `'update'` (the state at the end of the quarter, called after the RLS and the projection facility). The state lives in the learning state (`S.gain`), so every stage that carries the learning state carries the gain: the history, the inversion and its snapshots, the forecast, density draws, evaluation origins and the Ramsey loss continuation. With `'constant'` the gain is $g_{v,t} = \gamma_0$ (or the `gain.by_var` value of the equation). With `'decreasing'` it falls with the number of updates (Section 9.3).

9.2.1 `gain.rule`

Default `'constant'`. (Project baseline: `'koz'`.)

Values `'constant'`, `'koz'`, `'decreasing'`; the name must be a function in `+gain/`.

Math `'constant'`: $g_{v,t} = \gamma_0$ (or the `by_var` value). `'koz'`: $g_{v,t} = \gamma_{p(v),t-1}$, equations (20). `'decreasing'`: $g_{v,t} = 1/(t_0 + k_{v,t} + 1)$, equation (19).

Assumptions see the entries below.

Interactions `gain.value` is the constant gain and the KOZ start value, and is not used by `'decreasing'`; `gain.koz.*` matter only for `'koz'` and `gain.decreasing.t0` only for `'decreasing'`; with `beliefs.update = 'fixed'` no rule is called.

Code `+gain/constant.m`, `+gain/koz.m`, `+gain/decreasing.m`, `+engine/update_learning.m`.

Example

```
cfg.gain.rule = 'koz';
cfg.gain.value = 0.05;                          % start gain
```

9.2.2 `gain.value`

Default `0.0585`, the posterior mean of the constant gain in Özden (2025, Table 1). (Project baseline: 0.05 as the KOZ start.)

Values a number in $(0, 1]$.

Math γ_0 in (17)–(18) under `'constant'`; the initial KOZ gain when `gain.koz.gamma0` is empty.

Assumptions a constant gain weights past observations geometrically with factor $1 - \gamma_0$ (perpetual learning, tracking a drifting reduced form).

Interactions under `'koz'` it must lie inside `gain.koz.bounds`.

Code `+gain/constant.m`, `+gain/koz.m`.

Example

```
cfg.gain.rule = 'constant';
cfg.gain.value = 0.02;
```

9.2.3 `gain.by_var`

Default `struct()`.

Values `by_var.<target> = g` with $g \in (0, 1]$; `<target>` is a PLM target (a learning variable or an auxiliary one).

Math $g_{v,t} = g$ for that equation under `'constant'`; the start gain of its process under KOZ `'per_variable'`.

Assumptions learners of different variables may discount the past at different rates.

Interactions ignored with a warning (`hetlearn:gain:byVarIgnored`) under KOZ `'common'`. Free-form subtree: a name that is not a PLM target is ignored silently by the constant rule.

Code `+gain/constant.m`, `+gain/koz.m`.

Example

```
cfg.gain.by_var.pinf = 0.02;
```

9.3 The decreasing gain

With `gain.rule = 'decreasing'` the gain of equation v falls with the number of its updates (`+gain/decreasing.m`):

$$g_{v,t} = \frac{1}{t_0 + k_{v,t} + 1}, \tag{19}$$

where $k_{v,t}$ counts the updates of equation v up to and including quarter t ($k_{v,t} = 1$ in the first learning quarter, `beliefs.init.date`) and $t_0 =$ `gain.decreasing.t0`. The initial beliefs $(\phi_{v,0}, R_{v,0})$ count as $t_0 + 1$ observations, so with $t_0 = 0$ the gain is $1/t$ with t the number of observations including the initial beliefs: $1/2$ at the first update, $1/3$ at the second. The first update therefore averages $R_{v,0}$ and zz', and $R_{v,t}$ stays invertible for a PLM with several coefficients (a first gain of 1 would set $R_{v,1} = zz'$, which has rank one). With these gains the RLS recursion (17)–(18) gives least squares with equal weights on every observation and on the initial beliefs (Evans and Honkapohja, 2001, ch. 2); for a constant-only PLM, $R_{v,0} = 1$ and the belief is the running mean $\alpha_{v,t} = (\alpha_{v,0} + \sum_{s \le t} x_{v,s})/(k_{v,t} + 1)$ when $t_0 = 0$ (`test_gain` D3, D5). The gain goes to zero, so learning converges where the E-stability conditions hold, and beliefs move little late in a long history: from 1966Q1 the gain is about $1/227$ at 2022Q2.

A quarter in which an equation's data are missing (event `rls_missing`) does not count. The count is part of the gain state, so it carries on from the history into the forecast, density draws, evaluation origins and the Ramsey continuation, and it moves whatever the projection facility does with the candidate beliefs. With N agent types each learning type has its own count. `gain.value` and `gain.by_var` are not used.

9.3.1 `gain.decreasing.t0`

Default `0`.

Values a finite number ≥ 0.

Math t_0 in (19).

Assumptions the initial beliefs are worth t_0+1 observations; a larger t_0 gives them more weight and lowers every gain.

Interactions read only under `gain.rule = 'decreasing'`.

Code `+gain/decreasing.m`, `load_options.m`.

Example

```
cfg.gain.rule = 'decreasing';
cfg.gain.decreasing.t0 = 19;      % initial beliefs worth 20 quarters:
    first gain 1/21
```

9.4 The endogenous gain of Kostyshyna, Özden and Zhang

Kostyshyna et al. (2024, eq. 9) let the gain follow the adaptive step-size algorithm of Kostyshyna (2012) (see also Kushner and Yin, 2003). For each gain process p, after x_t is realized (`+gain/koz.m`):

$$\gamma_{p,t} = \Pi_{[\underline{\gamma},\overline{\gamma}]}\Big(\gamma_{p,t-1} + \mu\, R^g_{p,t-1}\, e_{p,t}\Big), \qquad (20)$$
$$R^g_{p,t} = (1-\gamma_{p,t-1})\, R^g_{p,t-1} + e_{p,t},$$

and the RLS update at the end of t uses $g_{v,t} = \gamma_{p(v),t-1}$, the gain of the process of equation v before this quarter's update. Π projects onto `gain.koz.bounds`; R^g is not projected. R^g tracks the derivative of the beliefs with respect to the gain: it adds up past errors with weight $1-\gamma$, so $R^g_{t-1}e_t > 0$ and the gain rises while the errors keep their sign, and the gain falls when they alternate.

The driving error is signed and scaled,

$$e_{p,t} = \frac{\sigma}{|D_p|}\sum_{d\in D_p} e_{d,t}, \qquad e_{d,t} = \begin{cases} v_{d,t} - \phi'_{d,t-1} z_{d,t-1} & \texttt{error = 'rls'} \\ x_{d,t} - f^{\mathrm{BR}}_{t-1}(x_d) & \texttt{error = 'forecast'} \end{cases} \qquad (21)$$

with $\sigma =$ `gain.koz.error_scale` and D_p the drivers of process p. A missing error (the first quarter under `'forecast'`, missing data, or a missing error of any driver of the list) leaves γ_p and R^g_p unchanged. A bound that binds sets γ to the bound and logs a `gain_bound` event; a non-finite update keeps the state and logs `gain_bound` too.

Timing. The `'gain'` call returns γ_{t-1} before the RLS candidates; the `'update'` call applies (20) after the RLS and the projection facility. A reverted or rejected belief update does not revert the gain: the error was observed either way. With $\mu = 0$, $\gamma_{p,t} = \gamma_{p,0}$ and the run equals the constant rule bit for bit (`test_gain` G2, G7).

Outputs. `res.learning.<target>.gain` is the gain used in t; `gain_R` is R^g_t of the target's process; `res.state.gain` is the final state. Constant-gain runs record no `gain_R`.

9.4.1 `gain.koz.mu`

Default `0.001`, the step size of Kostyshyna (2012) used by Kostyshyna et al. (2024).

Values a finite number ≥ 0.

Math μ in (20), the gain on the gain.

Assumptions μ is scale dependent: e enters both equations, so the effective step scales with σ^2.

Interactions `error_scale`; $\mu = 0$ reproduces `'constant'`.

Code `+gain/koz.m`.

Example

```
cfg.gain.rule = 'koz';
cfg.gain.koz.mu = 0.0005;
```

9.4.2 `gain.koz.gamma0`

Default `[]`: `gain.value`, and under `'per_variable'` `gain.by_var.<target>` where set.

Values `[]` or a number in $(0, 1]$ inside the bounds.

Math $\gamma_{p,0}$, the gain of the first RLS update.

Assumptions KOZ start the process at their estimated constant gain.

Interactions the initial gain (this value, or `gain.value` and the `by_var` values) must lie inside `gain.koz.bounds` (`hetlearn:options:value`).

Code `+gain/koz.m`, `load_options.m` (`validate_gain`).

Example

```
cfg.gain.rule = 'koz';
cfg.gain.koz.gamma0 = 0.03;
```

9.4.3 `gain.koz.Rg0`

Default `0`.

Values a finite number.

Math $R^g_{p,0}$ in (20). With $R^g_0 = 0$ the first update leaves the gain unchanged and sets $R^g_1 = e_1$.

Assumptions KOZ are silent; 0 states that the initial estimate does not depend on the gain. KOZ's Table 1 "$R_0 = 0.001$" is the RLS moment matrix, not R^g_0; it is set through `beliefs.init.values.<x>.R` (Section 10.1.3).

Interactions none beyond `'koz'`.

Code `+gain/koz.m`.

Example

```
cfg.gain.rule = 'koz';
cfg.gain.koz.Rg0 = 0;
```

9.4.4 `gain.koz.bounds`

Default `[0.001 0.25]`.

Values `[lower, upper]` with $0 <$ lower $\leq$ upper ≤ 1.

Math $[\underline{\gamma}, \overline{\gamma}]$ of the projection Π in (20).

Assumptions Kostyshyna (2012) projects the gain onto an interval; KOZ give none, so the interval is a toolbox choice.

Interactions the initial gain must lie inside; each binding quarter logs a `gain_bound` event (counted in density outputs).

Code Example `+gain/koz.m`.

```
cfg.gain.rule = 'koz';
cfg.gain.koz.bounds = [0.005 0.15];
```

9.4.5 `gain.koz.error`

Default `'rls'`.

Values `'rls'`, `'forecast'`.

Math the case in (21): the one-step PLM residual (the error of the RLS update itself, KOZ eq. 8) or the learners' expectation error $x_t - f_{t-1}^{\mathrm{BR}}(x)$.

Assumptions KOZ do not say which error; the residual is the default.

Interactions `'forecast'` is a two-step error under `beliefs.info = 'lagged'`; it is missing in the first learning quarter. Auxiliary equations have no expectation and use their residual. With `'common'` the master names PLM targets (`'rls'`) or learning variables (`'forecast'`).

Code Example `+gain/koz.m`, `+engine/update_learning.m`.

```
cfg.gain.rule = 'koz';
cfg.gain.koz.error = 'forecast';
```

9.4.6 `gain.koz.error_scale`

Default `1` (model units, percent per quarter for SW07).

Values a finite number > 0.

Math σ in (21).

Assumptions KOZ's data are annualized; a factor of 4 converts quarterly to annualized percent and acts like a 16 times larger μ.

Interactions `gain.koz.mu`.

Code Example `+gain/koz.m`.

```
cfg.gain.rule = 'koz';
cfg.gain.koz.error_scale = 4;              % annualized errors
```

9.4.7 `gain.koz.structure`

Default `'common'`.

Values `'common'`, `'per_variable'`.

Math `'common'`: one process for every PLM equation (auxiliary ones included), driven by the errors of `gain.koz.master` (Kostyshyna et al., 2024, fn. 7). `'per_variable'`: one process per PLM equation, driven by its own error.

Assumptions `'common'`: one learning speed for all beliefs.

Interactions `'common'` ignores `gain.by_var` (warning); `'per_variable'` ignores `gain.koz.master` (warning `hetlearn:gain:masterIgnored`) and takes start values from `gain.by_var`.

Code `+gain/koz.m` (`init`).

Example

```
cfg.gain.rule = 'koz';
cfg.gain.koz.structure = 'per_variable';
```

9.4.8 `gain.koz.master`

Default `''`: the first learning variable (the first PLM target if there is no learning variable).

Values a name or a cellstr of valid names.

Math D_p in (21): a list gives the mean of the scaled errors.

Assumptions the common learning speed reacts to the errors of the named variables.

Interactions `'common'` only. Under `'rls'` the names must be PLM targets, under `'forecast'` learning variables (`hetlearn:gain:master`, checked when the gain state is built). A missing error of any driver freezes the process that quarter.

Code `+gain/koz.m`.

Example

```
cfg.gain.rule = 'koz';
cfg.gain.koz.master = {'pinf', 'w'};
```

10 Initial beliefs

Initial beliefs (ϕ_0, R_0) are set once per run, by `+engine/state_init.m`, for every PLM equation (auxiliary ones included). They are the beliefs in force at $t =$ `beliefs.init.date`: they make the learners' forecast in that quarter, and the first RLS update uses that quarter's data.

10.1 Options

10.1.1 `beliefs.init.method`

Default `'re_moments'` (Özden, 2025, sec. 3.1).

Values `'re_moments'`, `'training_ols'`, `'user'`.

Math `'re_moments'` (`+init/re_moments.m`): for each equation,

$$R_{v,0} = E[z_v z_v'], \qquad \phi_{v,0} = E[z_v z_v']^{-1} E[z_v v], \tag{22}$$

the population projection of v_t on $z_{v,t-1}$ in the RE model. The moments are $E[a_{t-l_a} b_{t-l_b}] = \mathrm{Cov}(a_{t-l_a}, b_{t-l_b}) + \mu_a \mu_b$ with the autocovariances of IRIS `acf(m, 'order', max(maxlag, 1))` and the means μ from the steady state. With a constant, $E[zz'] = \left[\begin{smallmatrix} 1 & \mu' \\ \mu & E[uu'] \end{smallmatrix}\right]$. For a zero-mean AR(1) PLM this gives $\alpha_0 = 0$, $\beta_0 = \rho_1(x)$ and $R_0 = \mathrm{diag}(1, \mathrm{var}\, x)$. A near-singular $E[zz']$ (reciprocal condition below 10^{-14}) uses the pseudo-inverse with a warning. `'training_ols'` (`+init/training_ols.m`): OLS over the training range, $\phi_{v,0} = (Z'Z)^{-1}Z'y$, $R_{v,0} = Z'Z/T$, with rows $z_{v,t-1}'$ and targets v_t for t in the range; rows with a missing value are dropped. `'user'` (`+init/user.m`): the values given, the RE moments for anything not given.

Assumptions `'re_moments'`: learners start at the least-squares projection that the moments of the RE economy imply. The model is the loaded model of the run at $n_1 = 1$: the RE model of the scenario, the RE(t-1) model under `re_info = 'lagged'`, and the rule model under Ramsey policy. `'training_ols'`: learners start from a regression on the history they watch.

Interactions `'training_ols'` needs `beliefs.init.training_range` (`hetlearn:options:missing`) and reads the history databank of `history.re_states` (smoothed or filtered); fewer usable rows than coefficients is `hetlearn:init:shortSample`. `'user'` reads `beliefs.init.values`. With `history.learn = false` the initial beliefs apply at the origin.

Code `+engine/state_init.m` and, in `+init/`, `re_moments.m`, `training_ols.m`, `user.m`.

Example

```
cfg.beliefs.init.method = 'training_ols';
cfg.beliefs.init.training_range = [qq(1966,1), qq(1979,4)];
```

10.1.2 `beliefs.init.date`

Default `[]`: the first filter date plus 4 quarters (SW07: 1966Q1).

Values an IRIS date with room for the PLM lags after the filter start and, with `history.learn = true`, not after the origin (`hetlearn:engine:initDate`).

Math the quarter t_0 at which $\phi_{t_0-1} = \phi_0$ and the learning record starts.

Assumptions learning starts here; shares start at `shares.init_n2`.

Interactions the default start of `history.inversion.start`; the start of the Kalman likelihood (`loglik_start`) and of a `'smoothed'` Kalman start; ignored with `history.learn = false` (the record then starts at the origin).

Code `+engine/core.m`.

Example

```
cfg.beliefs.init.date = qq(1985,1);
cfg.history.learn = true;
```

10.1.3 `beliefs.init.values`

Default `struct()`.

Values `values.<target>` with any of `.phi` (the full vector, in parameter order), `.alpha`, `.beta` (the slope vector, regressor order), `.R` (a $k \times k$ matrix). `.phi` takes precedence over `.alpha`/`.beta`.

Math $\phi_{v,0}$, $R_{v,0}$ for the targets given; (22) for the rest.

Assumptions the user's values are the learners' prior beliefs.

Interactions `beliefs.init.method = 'user'` only. `.alpha` for a PLM without a constant and wrong sizes are errors (`hetlearn:init:noConstant`, `hetlearn:init:size`); entries for variables without a PLM equation are ignored with a warning. Setting only `.R` keeps the RE-moment ϕ and replaces R, e.g. KOZ's $R_0 = 0.001\,I$.

Code Example `+init/user.m`.

```
cfg.beliefs.init.method = 'user';
cfg.beliefs.init.values.pinf = struct('R', 0.001*eye(2));
```

10.1.4 `beliefs.init.training_range`

Default `[]`.

Values `[start, end]` or a date vector of target dates.

Math the set of t in the OLS of `'training_ols'`; regressors are read at $t - l_j$.

Assumptions the training sample is in the history the learners watch.

Interactions required by `'training_ols'`, unused otherwise. It can overlap the quarters of the learning record, in which case the learners start from beliefs fitted on data they later learn from again.

Code Example `+init/training_ols.m`.

```
cfg.beliefs.init.method = 'training_ols';
cfg.beliefs.init.training_range = [qq(1966,1), qq(1984,4)];
```

11 Switching between RE and learning

11.1 Forecast errors, fitness and shares

At the end of t the toolbox scores the two forecasts of x_t made in $t-1$ (`+engine/update_learning.m`):

$$FE^i_t(x) = \left(x_t - f^i_{t-1}(x)\right)^2, \qquad i \in \{\mathrm{RE}, \mathrm{BR}\}. \tag{23}$$

The squared errors enter unscaled, in model units. Each share group g sums them over its variables G_g, $FE^i_{g,t} = \sum_{v \in G_g} FE^i_t(v)$. Fitness has memory ω and the RE share follows a discrete choice with intensity χ (Özden, 2025, eqs. 14–15), (Brock and Hommes, 1997):

$$\zeta^i_{g,t} = -(1-\omega)\, FE^i_{g,t} + \omega\, \zeta^i_{g,t-1}, \tag{24}$$

$$n_{1,g,t} = \frac{\exp(\chi \zeta^{\mathrm{RE}}_{g,t})}{\exp(\chi \zeta^{\mathrm{RE}}_{g,t}) + \exp(\chi \zeta^{\mathrm{BR}}_{g,t})} = \frac{1}{1 + \exp\left(-\chi(\zeta^{\mathrm{RE}}_{g,t} - \zeta^{\mathrm{BR}}_{g,t})\right)}, \qquad n_{2,g,t} = 1 - n_{1,g,t}. \tag{25}$$

The logistic form is evaluated without overflow for any value of $\chi(\zeta^{\mathrm{RE}} - \zeta^{\mathrm{BR}})$ (`+switching/discrete_choice.m`). With $\omega = 0$ the memory term is dropped, so an infinite initial fitness does not produce $0 \cdot \infty$; $\chi = 0$ gives $n_1 = 1/2$; two infinite fitness values of the same sign give $1/2$. A non-finite share keeps the previous fitness and shares and logs a `shares_nan` event.

Timing. The shares computed at the end of t enter the model in $t+1$ (Özden, 2025, eq. 16): under `'switching'`, $n^m_{1,t+1} = n_{1,t}$ (Section 5.1.1). The forecasts scored at t were made in $t-1$, so the model in force in t did not see the errors of t.

Initial fitness. `+switching/init_fitness.m` sets

$$\zeta_0^{\mathrm{RE}} = 0, \qquad \zeta_0^{\mathrm{BR}} = \frac{\ln \bar{n}_2 - \ln(1 - \bar{n}_2)}{\chi}, \qquad n_{1,0} = 1 - \bar{n}_2, \tag{26}$$

with $\bar{n}_2$ = `shares.init_n2`, so the logit (25) returns $\bar{n}_2$ at the start. $\bar{n}_2 = 0$ gives $\zeta_0^{\mathrm{BR}} = -\infty$ and $\bar{n}_2 = 1$ gives $+\infty$. With $\chi = 0$ both fitness values are 0 and a warning says the shares will be $1/2$ after the first quarter.

Diagnostics outside switching. The fitness and the shares are updated in every mode; outside `'switching'` they are recorded (`n1`, `n2`, `zetaRE`, `zetaBR`) but do not enter the model, whose share is `n1_model`.

Zero future shocks. With zero shocks over the horizon both squared errors shrink, and the fitness of both predictors decays at rate ω; the shares drift toward $1/2$ by construction (there is no cost term for RE).

Units of χ. χ multiplies squared errors. Özden (2025) estimates $\chi = 0.51$ on data in percent per quarter, the units of SW07, so it applies as estimated; for annualized data the same switching needs $\chi/16$. A published intensity β of the unnormalized form $U_t = \omega U_{t-1} - FE_t$ corresponds to $\chi = \beta/(1-\omega)$ here.

11.2 Options

11.2.1 `switching.rule`

Default `'discrete_choice'`.

Values the name of a function `hetlearn.switching.<rule>` with the interface of `discrete_choice` (`[n1, zRE, zBR] = rule(FE_RE, FE_BR, zRE_prev, zBR_prev, chi, omega)`, vectorized over share groups).

Math equations (24)–(25).

Assumptions agents choose the predictor with the higher past fitness with probability given by a logit.

Interactions `switching.intensity`, `switching.memory`; the initial fitness always comes from `init_fitness`.

Code `+switching/discrete_choice.m`, `load_options.m` (the function must exist).

Example

```
cfg.switching.rule = 'discrete_choice';
```

11.2.2 `switching.intensity`

Default `0.51`, the estimate of Özden (2025) on quarterly data. (Project baseline: 2.)

Values a number ≥ 0.

Math χ in (25) and (26).

Assumptions $\chi = 0$: shares stay at 1/2 after the first quarter; $\chi \to \infty$: everyone takes the better predictor.

Interactions the units of the squared errors (above); `shares.init_n2` through (26).

Code Example `+switching/discrete_choice.m`, `+switching/init_fitness.m`.

```
cfg.switching.intensity = 1;
cfg.switching.memory = 0.6;
```

11.2.3 `switching.memory`

Default `0.6` (Özden, 2025, sec. 4; the Table 1 posterior mean is 0.62). (Project baseline: 0.8.)

Values a number in $[0, 1)$.

Math ω in (24).

Assumptions fitness is an exponentially weighted average of past squared errors with weight $1 - \omega$ on the newest.

Interactions $\omega = 0$ drops the memory term.

Code Example `+switching/discrete_choice.m`.

```
cfg.switching.memory = 0.8;
```

11.2.4 `shares.init_n2`

Default `0.5`.

Values a number in $[0, 1]$.

Math $\bar{n}_2$ in (26): the learners' share at `beliefs.init.date` (and at the origin with `history.learn = false`).

Assumptions the initial fitness difference is chosen to return this share.

Interactions used by `'switching'` through the fitness; under `'switching'` it is also the model share in the first forecast quarter when `history.learn = false`. The other modes ignore it for the model share.

Code Example `+engine/state_init.m`, `+switching/init_fitness.m`.

```
cfg.shares.init_n2 = 0.2;
cfg.expectations.mode = 'switching';
```

11.2.5 `shares.fixed_n2`

Default `[]`.

Values a number in $[0, 1]$, or a struct by learning variable with values in $[0, 1]$.

Math $\bar{n}_2$ in (4): $n^m_{1,t} = 1 - \bar{n}_2$ (or $1 - \bar{n}_{2,x}$) in every quarter.

Assumptions the population is fixed; switching is a diagnostic.

Interactions required by `expectations.mode = 'fixed_shares'` (`hetlearn:options:missing`); a struct must have an entry for every learning variable (`hetlearn:engine:fixedShares`). Not together with the N-type `shares.fixed`. Fixed shares of 0 make the ALM check skip.

Code `+engine/shares_for_mode.m`.

Example

```
cfg.expectations.mode = 'fixed_shares';
cfg.shares.fixed_n2 = struct('pinf', 0.3, 'w', 0.5);
```

11.2.6 `shares.structure`

Default `'per_variable'`.

Values `'per_variable'`, `'universal'`.

Math `'per_variable'`: one group per learning variable, $G_x = \{x\}$, each with its own fitness and share. `'universal'`: one group for all learning variables, $G =$ `shares.master`; every learning variable uses the group's share.

Assumptions `'universal'`: an agent is RE or a learner for every variable at once, and judges predictors by the summed squared errors of the master variables, as Özden (2025) does for inflation and the output gap.

Interactions `'universal'` needs `shares.master` (`hetlearn:options:missing`). The recorded `FE_RE`, `FE_BR` stay each variable's own errors; the group sums drive the shares.

Code `+engine/state_init.m` (share groups), `+engine/update_learning.m`.

Example

```
cfg.shares.structure = 'universal';
cfg.shares.master = {'pinf', 'y'};
```

11.2.7 `shares.master`

Default `{}`.

Values a name or a cellstr of learning variables.

Math G of the universal group: $FE^i_t = \sum_{v \in G} FE^i_t(v)$.

Assumptions the share responds to the errors of these variables only.

Interactions `'universal'` only; every name must be a learning variable (`hetlearn:engine:master`).

Code `+engine/state_init.m`.

Example

```
cfg.shares.structure = 'universal';
cfg.shares.master = 'pinf';
```

12 The projection facility

Learning models can lose their stable solution. The projection facility is the toolbox's only stabilizing device (the earlier devices, partial updates, shrinking, a root threshold and error scaling, were removed; their option names raise `hetlearn:options:removed`). It restricts beliefs in the spirit of Marcet and Sargent (1989) and Evans and Honkapohja (2001).

12.1 PLM blocks and their roots

Two PLM equations are linked when one has the other's target among its regressors (any lag). A *block* is a connected component of that graph (`+plm/blocks.m`): a VAR is one block; an auxiliary equation belongs to the block of the equations that use it, so two learning variables that share a regressor are one block; separate AR(1) PLMs are separate blocks. A block is closed, so its companion matrix is defined (`+plm/companion.m`): with y_t the n targets of the block and L its largest lag,

$$X_t = FX_{t-1}, \qquad X_t = (1,\, y_t',\, y_{t-1}',\, \ldots,\, y_{t-L+1}')', \qquad \rho = \max\left|\mathrm{eig}(F_{2:,2:})\right|, \tag{27}$$

the spectral radius without the unit root of the constant ($\rho = |\beta|$ for an AR(1); NaN if a coefficient is not finite; 0 for a mean-only PLM). The companion of the whole system is block diagonal after a permutation, so its largest root is the largest block root.

12.2 The three settings

12.2.1 `projection.facility`

Default `'plm_alm'`. (Project baseline: `'alm'`.)

Values `'plm_alm'`, `'alm'`, `'none'`.

Math at the end of t, after the RLS candidates $(\tilde{\phi}_t, \tilde{R}_t)$ are formed from (ϕ_{t-1}, R_{t-1}) (`+engine/facility.m`):

(a) *PLM part* (`'plm_alm'` only). For each block with an updated equation, compute ρ of the block at the candidate beliefs (the other equations of the block at their previous values). If ρ is finite, $\rho \leq 1$ and every candidate R is finite, the block takes its candidates; otherwise all its equations keep ϕ_{t-1} and R_{t-1} (event `projection_plm`, with the block and its root). Every candidate comes from the previous beliefs and each block is judged on its own equations, so the result does not depend on the equation order.

(b) *ALM check* (`'alm'` and `'plm_alm'`). Solve the model of $t+1$: beliefs ϕ_t and the mode's shares for $t+1$. It is stable if IRIS finds a unique stable solution, or, for a model without leads, if the backward system's largest root is at most 1. The check is skipped when every learners' model share is 0.

(c) *Revert.* If the ALM is unstable and this quarter changed any belief, every belief (ϕ and R of every PLM equation, auxiliary ones included) goes back to its value before this quarter's update, (ϕ_{t-1}, R_{t-1}), and the model is solved again (event `projection_alm`). There is no search for the equation that causes the instability.

(d) *Keep the shares.* If the model is still unstable and the new model shares differ from those in force in t, the shares in force in t are kept for $t+1$ with the reverted beliefs, i.e. the model of t, which solved (event `projection_alm_shares`; state `S.n1m_keep`, one quarter only). The fitness and the discrete-choice shares $n_{1,t}$ keep their update; the next quarter tries the new shares again.

(e) *Stop.* If no step gives a stable model, the run stops with `hetlearn:engine:almUnstable` (date, stage, shares, beliefs, IRIS flag). This happens only when no earlier solved model exists: the initial beliefs and shares are unstable at the first check of a run, or a stage checks a model other than the one checked before (the Ramsey forecast model at the origin) and the previous beliefs and shares do not solve it either.

`'none'` restricts nothing: no PLM part and no ALM check. A quarter that must be simulated without a solution stops with `hetlearn:engine:noSolution` (date, stage, learners' shares, the largest root of every PLM block, beliefs, IRIS flag), except a linear model without leads, which runs on its backward system with `explosive` events.

Assumptions `'alm'` allows explosive PLMs as long as the economy is stable; `'plm_alm'` also rules out explosive perceived dynamics block by block. Beliefs and shares last in force are always admissible because they solved.

Interactions the facility never touches the gain state: a KOZ gain updates from the observed error whatever the facility did. A revert detected at the start of a quarter (a model that the end-of-quarter check did not solve, e.g. the first forecast quarter with `history.learn = false`) rewrites the previous quarter's record, so `res.learning` holds the beliefs actually in force; kept shares become that quarter's `n1_model`. With `beliefs.update = 'fixed'` the revert step finds nothing to revert and goes to the share step. With fixed shares the share step finds nothing to keep. Under `'none'` a comparison target that stops does not fail the run that compares with it (warning `hetlearn:save_results:compare`).

Code `+engine/facility.m`, `+plm/blocks.m`, `+plm/companion.m`, `+engine/update_learning.m`, `+engine/shares_for_mode.m`, `+engine/solve_model.m`.

Example

```
cfg.projection.facility = 'alm';            % restrict the ALM only
cfg.expectations.variables = 'all';
```

12.3 Kept shares

The kept shares (step d) apply for one quarter: `+engine/update_learning.m` clears `S.n1m_keep` before the next check and records the shares in force in t as `S.n1m_last`. In the record, `n1`, `n2` are the discrete-choice shares and `n1_model` the share in force, kept ones included. Over the RE-smoothed history `n1_model` stays NaN (no model is simulated there), but kept shares are carried in the state, so the next check and the first forecast quarter use them.

12.4 Where the check runs

Beliefs formed at the end of t are in force in $t+1$, so the ALM check at the end of t solves the model of $t+1$, and the stage reuses that solution in $t+1$ without another solve. The same rules apply in every stage that updates beliefs:

Stage	Model checked at the end of t
RE-smoothed history	the history model before the origin (the rule model of a Ramsey scenario); the forecast model at the origin, whose solution the forecast's first quarter reuses (`fopt.warm`). The model is solved only for the check
Inversion history	the model being inverted; the forecast model at the origin
Kalman history	the ALM being filtered; the forecast model at the origin
Forecast	the model of the next quarter
Density draws, evaluation runs, Ramsey loss continuation	as the forecast

`res.meta.history_alm_check_s`, `.history_alm_n_check` and `.history_alm_n_solve` measure the history-stage checks; `res.meta.forecast_n_solve` counts the forecast's solves and `res.meta.projection_facility` records the setting.

12.5 Events

Event	Meaning
`projection_plm`	a PLM block's candidate had a root above 1 (or was not finite); the block kept its beliefs
`projection_alm`	the ALM with the new beliefs had no stable solution; every belief was reverted
`projection_alm_shares`	still unstable with the old beliefs; the previous quarter's model shares stayed in force
`explosive`	a model without leads was simulated by the backward system (no stable IRIS solution)
`rls_missing`	missing data in a PLM equation; it kept its beliefs
`shares_nan`	non-finite shares; the previous fitness and shares were kept
`gain_bound`	the KOZ gain hit a bound of `gain.koz.bounds` (or a non-finite gain update was skipped)

Events are returned in `res.events` and `events.csv` with columns `date`, `type`, `variable`, `detail` and `stage` (`history` up to the origin, `inversion` from `history.inversion.start` to the origin in an inversion run, `forecast` after the origin; `+engine/assemble.m`).

13 The history stage

The history stage runs from `beliefs.init.date` t_0 to the origin T_0. It produces the learning state at the end of T_0 (beliefs, moments, fitness, shares, gain state, the forecasts of x_{T_0+1} made at T_0) and the databank the forecast starts from. Four designs are available through `history.mode` and `history.re_states`:

Setting	What the agents watch	What the forecast starts from
`'re_smoother'`, `'smoothed'`	the RE-smoothed history (two-sided)	the RE-smoothed state at T_0
`'re_smoother'`, `'filtered'`	the RE-filtered history (one-sided)	the RE-smoothed state at T_0
`'inversion'`	the history backed out with the learning model	the backed-out state at T_0
`'kalman'`	the real-time filtered state of the ALM in force	the filtered state at T_0

Under `'re_smoother'` the path of the economy does not depend on the learning state: agents learn while watching the RE economy. Under `'inversion'` and `'kalman'` the learning model generates the history.

13.1 The RE filter and smoother

`hetlearn.re_filter(spec, opts)` (`re_filter.m`) runs the IRIS Kalman filter and smoother of the RE model over `spec.filter_range` and caches the result.

The filter model. Under `expectations.re_info = 'current'` it is the extended model with every `br_x = true` (`expectations.variables = 'all'`), $n_1 = 1$, zero beliefs, the scenario's `re_variant` and `beliefs.info`, and the policy rule: the RE model, with every `x_exp1_RE` in the smoothed history. Under `'lagged'` it keeps the scenario's learning set (the RE(t-1) model of $\mathcal{L}$). Ramsey scenarios are always filtered with the rule model.

Outputs. One call `filter(m, d, range, 'output', 'filter,smooth', 'meanOnly', true)` (`'pred,filter,smooth'` for lagged models) gives the databanks the stages read:

- `H.smooth`: $x_{t|T}$, two-sided, every transition and measurement variable and shock;
- `H.filtered`: $x_{t|t}$, one-sided (data up to t);
- `H.predicted`: $x_{t|t-1}$, lagged models only.

The other fields of `H` hold the database (clipped in real time), the range, the fit error and the cache information. Observed measurement variables equal the data in every output; where every observable is missing, the filtered value is the one-step prediction and the filtered shocks are 0.

Fit check. The observables are rebuilt from the smoothed states and shocks through the measurement equations (`lhsmrhs` with the data on the left). With at least as many shocks as observables the largest absolute error must be below 10^{-8} (`hetlearn:re_filter:fit`). Comparing the smoothed observables with the data would be vacuous, since IRIS copies the data into them.

Cache. `generated/refilter_<hash>.mat`, with the hash over the extended model text, the parameter values, the assign struct (toggles), the database file bytes and the filter range; in real time `refilter_rt_<date>_<hash>.mat`, with the clipped data values in place of the file bytes, so a data change after the end date does not touch it. A cache that lacks the filtered databank, or a lagged cache without predictions, is filtered again and rewritten.

13.2 Learning on the RE history

With `history.mode = 're_smoother'` and `history.learn = true`, the sequence of Section 2.2 runs for $t = t_0, \ldots, T_0$ on the history databank D of `history.re_states` (`+engine/history.m`, `+engine/history_db.m`), with three differences. Steps 2 and 3 are skipped: the model is never simulated, and it is solved only for the ALM check of step 5 under `'alm'` and `'plm_alm'` (Section 12.4). In step 4 the forecasts come from D: $f_t^{\mathrm{RE}}(x) = D.\texttt{x_exp1_RE}(t)$, and $f_t^{\mathrm{BR}}(x) = \mathcal{E}(x, 1)$ of (10) with ϕ_{t-1} and the data $D.u(t+k)$, $k \leq k_{\max}$. In step 5 the realized value is $x_t = D.x(t)$. `res.paths.x_exp1_BR` holds the learners' forecasts over the record dates (NaN before t_0), not the zero-belief values of the RE filter.

13.2.1 `history.mode`

Default `'re_smoother'`.

Values `'re_smoother'`, `'inversion'`, `'kalman'`. `'hetero_filter'` is a reserved name (`hetlearn:options:notImplemented`, pointing to `'kalman'`).

Math Sections 13.2, 13.3, 13.4.

Assumptions `'re_smoother'`: the data are RE data, the agents' learning does not feed back into the history. `'inversion'`: the learning model explains the data exactly through the endogenized shocks. `'kalman'`: the learning model explains the data up to filtering uncertainty, quarter by quarter.

Interactions `'inversion'` and `'kalman'` need `history.learn = true` and `history.re_states = 'smoothed'`; `history.realtime` applies to `'re_smoother'` only (Section 13.2.3); `hetlearn.decompose` refuses `'kalman'` results; `hetlearn.evaluate` runs one inversion per inversion variant.

Code `+engine/core.m`, `+engine/history.m`, `+engine/kalman_history.m`.

Example

```
cfg.history.mode = 'inversion';
cfg.history.re_states = 'smoothed';
```

13.2.2 `history.re_states`

Default `'smoothed'`. (Project baseline: `'filtered'`.)

Values `'smoothed'`, `'filtered'`.

Math $D =$ `H.smooth` or `H.filtered`. Under `'filtered'` $f_t^{\mathrm{RE}}(x) = x^{\mathrm{RE}}_{t+1|t}$, the filter's one-step prediction of the RE model, and FE_t^{RE} is its squared one-step prediction error; the learners' data and the realized x_t are $x_{t|t}$. With `re_info = 'lagged'` the RE forecasts come from `H.predicted` instead (Section 7.2).

Assumptions `'smoothed'`: the RE forecast at t uses states estimated with data after t, which gives RE agents hindsight in the fitness comparison. `'filtered'` removes it. For an observed variable the two coincide (filtered = smoothed = data); in a quarter without data the filtered value is the RE prediction, so $FE_t^{\mathrm{RE}} = 0$ there.

Interactions `'filtered'` with `'inversion'` or `'kalman'` is an error (`hetlearn:options:value`): both are one-sided already. It also sets the data of `history.learn = false` and of `beliefs.init.method = 'training_ols'`. The forecast still

starts from the smoothed state at the origin; with `history.realtime = true` that state is real time too, which makes `'filtered'` plus `realtime` the fully real-time setup. The filtered values up to the origin do not depend on later data, so `realtime` does not change the history loop under `'filtered'`.

Code Example `+engine/history_db.m`, `re_filter.m`.

```
cfg.history.re_states = 'filtered';
cfg.history.realtime = true;                % fully real time
```

13.2.3 `history.realtime`

Default `false`. (`hetlearn.evaluate` sets it from `opts.realtime`, default true.)

Values true, false.

Math with true, `hetlearn.run` calls `hetlearn.re_filter(..., 'end', T0)`: every database series is clipped at T_0 and the filter and smoother run over [filter start, T_0], so $x_{t|T}$ becomes $x_{t|T_0}$.

Assumptions no data after the origin reach the history or the starting state.

Interactions applies only with `history.mode = 're_smoother'` (`run.m`); under `'inversion'` (one-sided by construction, the full-sample smoother supplies only the state before the inversion start) and `'kalman'` the full-sample filter is used and the option has no effect. Real-time caches are kept per end date.

Code Example `run.m`, `re_filter.m` (`clip_data`).

```
cfg.history.realtime = true;
cfg.forecast.origin = qq(2015,1);
```

13.2.4 `history.learn`

Default `true`.

Values true, false.

Math false: no history loop; $\phi_{T_0} = \phi_0$, $R_{T_0} = R_0$, $n_{1,T_0} = 1 - \bar{n}_2$, and the forecasts made at T_0 come from the same formulas at $t = T_0$ on the history databank. No update at the origin.

Assumptions the initial beliefs and shares apply directly at the origin.

Interactions the record starts at T_0; `beliefs.init.date` is not used; `'inversion'` and `'kalman'` are errors. The first forecast quarter is the first ALM check, so an unstable initial model stops with `hetlearn:engine:almUnstable`.

Code Example `+engine/core.m`, `+engine/history.m`.

```
cfg.history.learn = false;
cfg.beliefs.init.method = 'user';
```

13.3 The inversion history

With `history.mode = 'inversion'` the history is simulated one quarter at a time with the learning model, as a forecast with conditions (`+engine/history.m`, function `inversion`). Let t_s = `history.inversion.start`. Before t_s the state is the RE-smoothed one; if $t_s > t_0$ the learners first learn on the RE-smoothed history over $[t_0, t_s - 1]$ (Section 13.2). For $t = t_s, \ldots, T_0$ the sequence of Section 2.2 runs (stage `'inversion'`) with one difference: the simulation of step 3 exogenizes every observable that has data at t at its data value and endogenizes the shocks of `history.inversion.shocks`. With y_t^0 the observables of the simulation with zero shocks and M their response to the endogenized shocks,

$$M\delta = y_t^{\text{obs}} - y_t^0, \tag{28}$$

solved exactly (as many shocks as observables) or with minimum norm in standard-deviation units (more shocks; Section 14.4); observables without data at t are not exogenized. Every variable at t is stored. The update at t_s is the first-quarter update of Section 9.1 only when $t_s = t_0$.

Quarter t uses data up to t, so the inversion is one-sided by construction. The forecast starts from the backed-out history: `+engine/core.m` replaces the RE-smoothed variables and shocks over $[t_s, T_0]$ by the backed-out ones. Data come from `H.data`. Under mode `'rational'` the inversion from the smoothed state reproduces the RE-smoothed shocks (test target 10^{-8}; 2×10^{-13} on SW07 over 1966Q1–2026Q2 when every quarter has data). In a quarter without data nothing is exogenized and the endogenized shocks stay 0, so from such a quarter on the inversion departs from the smoother, which infers shocks there from later data. A Ramsey scenario backs out its history with the rule model (untested).

Diagnostics (`res.inversion`). `shocks` holds the endogenized shocks and `params` the parameter values in the model each quarter, which `hetlearn.decompose` uses. `shock_diff_vs_smoothed` and `path_diff_vs_smoothed` give the largest absolute distance from the RE smoother (`x_exp1_BR` excluded), and `max_obs_miss` the largest miss of an observable target. The other fields hold dates, counts and run times.

13.3.1 `history.inversion.start`

Default `[]`: `beliefs.init.date`.

Values a date in $[t_0, T_0]$ (`hetlearn:engine:inversion` otherwise).

Math t_s above.

Assumptions before t_s the economy is the RE-smoothed one.

Interactions `'inversion'` only; with $t_s > t_0$ the learners learn on the smoothed history first; the events after t_s carry the stage `inversion`.

Code `+engine/core.m`, `+engine/history.m`.

Example

```
cfg.history.mode = 'inversion';
cfg.history.inversion.start = qq(1985,1);
```

13.3.2 `history.inversion.shocks`

Default `{}`: every transition shock of the model.

Values	a cellstr of shocks; at least as many as observables.
Math	the shocks whose values δ solve (28).
Assumptions	every data surprise is attributed to these shocks; the others stay 0.
Interactions	fewer shocks than observables or an unknown name is `hetlearn:engine:inversion`; a rank-deficient response is `hetlearn:engine:conditions`; with more shocks than observables `engine.conditions_weighting` sets the norm.
Code	`+engine/history.m`, `+engine/step.m`.

Example

```
cfg.history.mode = 'inversion';
cfg.history.inversion.shocks = {'ea', 'eb', 'eg', 'eqs', 'em', 'epinf', '
    ew'};
```

13.4 The Kalman history

With `history.mode = 'kalman'` the history is a time-varying Kalman filter of the ALM in force each quarter (see Durbin and Koopman, 2012, for the filter); beliefs and shares move with the same update as everywhere else (`+engine/kalman_history.m`). For quarter t from the start date to T_0:

(1) The ALM in force. Before t_0 the RE limit ($n_1 = 1$); from t_0 the model of steps 1 and 2 of Section 2.2 (stage `'kalman'`). Its state space on the full transition vector $\xi = (x^f, x^b)$ is

$$\xi_t = A\xi_{t-1} + \mathbf{K} + \mathbf{R}\varepsilon_t, \quad A = [0,\ \mathbf{T}], \qquad y_t = Z_f\xi_t + \mathbf{H}\varepsilon_t + \mathbf{D}, \quad Z_f = [0,\ \mathbf{Z}], \qquad \mathrm{Cov}(\varepsilon_t) = \Omega, \tag{29}$$

with Ω from the model's `std_*` and `corr_*` parameters (a backward system is used the same way).

(2) Prediction and update. With $G = Z_f\mathbf{R} + \mathbf{H}$,

$$\begin{aligned} a_{t|t-1} &= Aa_{t-1|t-1} + \mathbf{K}, & P_{t|t-1} &= AP_{t-1|t-1}A' + \mathbf{R}\Omega\mathbf{R}', \\ \hat{y}_{t|t-1} &= Z_f a_{t|t-1} + \mathbf{D}, & C_t &= P_{t|t-1}Z_f' + \mathbf{R}\Omega\mathbf{H}', \\ F_t &= Z_f C_t + \mathbf{H}\Omega G', & v_t &= y_t - \hat{y}_{t|t-1}, \end{aligned} \tag{30}$$

and, on the rows o with data,

$$\ell_t = -\tfrac{1}{2}\Big(n_t \ln 2\pi + \ln\det F_{oo,t} + v_{o,t}'F_{oo,t}^{-1}v_{o,t}\Big), \tag{31}$$

$$a_{t|t} = a_{t|t-1} + C_{o,t}F_{oo,t}^{-1}v_{o,t}, \quad P_{t|t} = P_{t|t-1} - C_{o,t}F_{oo,t}^{-1}C_{o,t}', \quad \varepsilon_{t|t} = \Omega G_o'F_{oo,t}^{-1}v_{o,t}. \tag{32}$$

A quarter without data is a prediction only, with $\ell_t = 0$. A covariance F_{oo} that is not positive definite stops the run. The log likelihood is $\sum_{t\geq t_0}\ell_t$.

(3) The agents' information (from t_0). RE forecasts made with data to $t-1$ (`re_info = 'lagged'`) are the predictions $\texttt{x_exp1_RE}_{t|t-1}$; with data up to t the filtered $\texttt{x_exp1_RE}_{t|t}$. The learners' PLM forecasts (10), their regressors and the realized values are the filtered values $x_{s|s}$ stored when each quarter was filtered, in real time, not revised later.

(4) Update. Step 5 of Section 2.2, with the first-quarter update at t_0.

The forecast starts from the filtered state at T_0: the lags of the state vector at the origin take their values in $a_{T_0|T_0}$.

Outputs (`res.kalman`).

Field	Content
`dates`, `observables`	the filtered quarters and the measurement variables
`y`, `pred`, `err`, `F`	data y_t, predictions $\hat{y}_{t\|t-1}$, errors v_t, covariances F_t
`llt`, `n_obs`	ℓ_t and the number of observables with data, by quarter
`loglik`, `loglik_start`	$\sum_{t \geq t_0} \ell_t$ and its first quarter t_0
`filtered`, `predicted`	databanks of $x_{t\|t}$ and $x_{t\|t-1}$
`state_names`, `a_origin`, `P_origin`	the state vector ξ, $a_{T_0\|T_0}$ and $P_{T_0\|T_0}$
`fbr_model_maxdiff`	the largest gap between the learners' forecast and the model's `x_exp1_BR` (below)
`n_solve`, `runtime_s`	solves and run time

`fbr_model_maxdiff` compares f_t^{BR} on the real-time data with the model's `x_exp1_BR` (predicted under `'lagged'`, filtered under `'current'`). The gap is rounding for a PLM whose forecast uses data of one date only, such as an AR(1); for an AR(2) it is not, because the model's lagged states are the filter's revised values and the agents' data the real-time ones. `hetlearn.run` writes `kalman_errors.csv` (date, `n_obs`, `llt`, and `err_`, `pred_`, `var_` of every measurement variable); `hetlearn.engine.kalman_history('rmse', res, subsamples)` gives one-step RMSEs and log likelihoods by subsample.

13.4.1 `history.kalman.init`

Default `'stochastic'`.

Values `'stochastic'`, `'smoothed'`.

Math `'stochastic'`: the filter starts at the filter start date from the asymptotic distribution of the first quarter's ALM (the RE limit), as IRIS `filter` does: $a_0 = (I - \mathbf{T}_b)^{-1}\mathbf{K}_b$, P_0 the solution of $P = \mathbf{T}_b P \mathbf{T}_b' + Q_{bb}$ (stationary roots only: a root of modulus $\geq 1 - 10^{-10}$ is an error). `'smoothed'`: the filter starts at t_0 from the RE-smoothed state of $t_0 - 1$ with $P_0 = 0$; the agents' data before t_0 are RE-smoothed.

Assumptions `'smoothed'` with as many shocks as observables and no measurement error makes every update exact, so the filter equals the inversion until a quarter without data.

Interactions `'kalman'` only; needs `history.learn = true` and `history.re_states = 'smoothed'`; under `projection.facility = 'none'` a quarter without a model solution stops with `hetlearn:engine:noSolution`.

Code `+engine/kalman_history.m` (`initial_state`).

Example

```
cfg.history.mode = 'kalman';
cfg.history.kalman.init = 'smoothed';
```

14 The forecast stage

14.1 The quarter-by-quarter loop

`+engine/forecast.m` runs the sequence of Section 2.2 for $t = T_0 + 1, \ldots, T_0 + H$, starting from the learning state at the end of T_0 and the databank of `+engine/build_shocks.m`: the history up to T_0 (RE-smoothed, or backed out, or filtered, depending on `history.mode`) and the scenario's shocks over the horizon. After each quarter the simulated values are written into the databank, so quarter $t + 1$ starts from the scenario's own t. The result is an *unconditional* forecast when the shocks are zero and there are no conditions.

Three properties hold by construction and are tested:

- with mode `'rational'` the forecast equals one IRIS `simulate` of the RE model over the horizon with unanticipated shocks (`test_engine` E1);
- with mode `'rational'`, `shocks.source = 'smoothed'` and the smoothed starting state, the forecast reproduces the data (`test_sw07`);
- the learners' forecast read from the model equals the PLM formula with ϕ_{t-1} in every quarter (the timing check).

14.2 Origin and horizon

14.2.1 `forecast.origin`

Default `[]` (must be set: `hetlearn:engine:origin`).

Values an IRIS date inside the filter range, with room for the PLM lags after the filter start.

Math T_0, the last quarter of data used; the forecast covers $T_0 + 1, \ldots, T_0 + H$.

Assumptions without `history.realtime` the smoothed state at T_0 uses the whole sample, including data after T_0.

Interactions `beliefs.init.date` must not be later (with learning); `history.inversion.start` must not be later; `hetlearn.evaluate` overrides it.

Code Example `+engine/core.m`.

```
cfg.forecast.origin = qq(2022,2);
cfg.forecast.horizon = 16;
```

14.2.2 `forecast.horizon`

Default `16`.

Values a positive integer.

Math H.

Assumptions none.

Interactions conditions and `shocks.add` dates must lie inside the horizon (dates of `shocks.range` and `density.range` outside it are ignored); `policy.loss_horizon` beyond H triggers a continuation; `hetlearn.evaluate` sets it from `opts.horizon`.

Code `+engine/core.m`.

Example

```
cfg.forecast.horizon = 8;
cfg.policy.loss_horizon = 80;
```

14.3 Shocks over the horizon

Every shock of the model is set to 0 over the horizon, then `shocks.source` and `shocks.add` fill it (`+engine/build_shocks.m`). All shocks are unanticipated.

14.3.1 `shocks.source`

Default `'none'`.

Values `'none'`, `'smoothed'`.

Math `'none'`: $\varepsilon_t = 0$. `'smoothed'`: $\varepsilon_t = \varepsilon_{t|T}$, the RE-smoothed shocks (`H.smooth`) over `shocks.range`, missing values set to 0.

Assumptions `'smoothed'` replays the shocks the RE model attributes to the data. They are the RE-smoothed shocks under every history mode, not the backed-out ones.

Interactions `shocks.range`, `shocks.exclude`; under Ramsey policy smoothed rule shocks run with no effect (warning `hetlearn:policy:ruleInputs`).

Code `+engine/build_shocks.m`.

Example

```
cfg.shocks.source = 'smoothed';
cfg.shocks.exclude = {'em'};
```

14.3.2 `shocks.range`

Default `[]`: the horizon.

Values a date vector, or `[start, end]`.

Math the quarters that receive smoothed shocks; dates outside the horizon are ignored.

Assumptions none.

Interactions `shocks.source = 'smoothed'` only.

Code `+engine/build_shocks.m`, `+util/to_dates.m`.

Example

```
cfg.shocks.source = 'smoothed';
cfg.shocks.range = [qq(2022,3), qq(2023,4)];
```

14.3.3 `shocks.exclude`

Default `{}`.

Values a cellstr of shocks of the model (an unknown name is `hetlearn:engine:shocks`).

Math the listed smoothed shocks are set to 0.

Assumptions a counterfactual without those shocks.

Interactions `shocks.source = 'smoothed'`; `shocks.add` still adds to an excluded shock.

Code `+engine/build_shocks.m`.

Example

```
cfg.shocks.source = 'smoothed';
cfg.shocks.exclude = {'epinf', 'ew'};
```

14.3.4 `shocks.add`

Default `{}`.

Values a cell array of triples `{name, dates, values}`: a shock, a date vector or `[start, end]`, and a scalar or one value per date.

Math $\varepsilon_{j,t} \leftarrow \varepsilon_{j,t} + v_t$ at the given dates.

Assumptions each added shock is a surprise in its quarter.

Interactions adds to the source; under Ramsey a rule shock runs with no effect. Free-form subtree (checked by `build_shocks`: `hetlearn:engine:shocks`).

Code `+engine/build_shocks.m`.

Example

```
cfg.shocks.add = {{'em', qq(2022,3), 0.25}, ...
                  {'epinf', [qq(2022,3), qq(2022,4)], [0.1 0.05]}};
```

14.4 Conditions: exogenize and endogenize

A condition puts a variable on a path in given quarters by endogenizing a shock. In quarter t let $\mathcal{K}_t$ be the conditioned variables with targets τ_t, $\mathcal{J}_t$ the endogenized shocks, s_t^0 the values of the conditioned variables in the one-period simulation with the scheduled shocks, and M_t their response to a unit of each endogenized shock. The engine solves (`+engine/step.m`)

$$M_t\,\delta_t = \tau_t - s_t^0, \qquad \delta_t = \begin{cases} M_t^{-1}(\tau_t - s_t^0) & |\mathcal{J}_t| = |\mathcal{K}_t| \\ S\,(M_t S)^{+}(\tau_t - s_t^0) & |\mathcal{J}_t| > |\mathcal{K}_t|, \ \texttt{'std'} \\ M_t^{+}(\tau_t - s_t^0) & |\mathcal{J}_t| > |\mathcal{K}_t|, \ \texttt{'none'} \end{cases} \tag{33}$$

with $S = \mathrm{diag}(\sigma_j)$, the standard deviations `std_<shock>`; the `'std'` case is the minimum of $\sum_j (\delta_j/\sigma_j)^2$ subject to the targets. Then $\varepsilon_{j,t} \leftarrow \varepsilon_{j,t} + \delta_{j,t}$ and the quarter is simulated again; every target must be met within 10^{-9}. Fewer shocks than targets, or $\mathrm{rank}\, M_t < |\mathcal{K}_t|$, is `hetlearn:engine:conditions`. Under `engine.step = 'iris'` M_t comes from unit-shock simulations; under `'fast'` it is exact: the rows of $\mathbf{R}$ for transition variables and $\mathbf{Z}\mathbf{R}_b + \mathbf{H}$ for measurement variables. The procedure equals an IRIS `Plan` with exogenize/endogenize pairs run one quarter at a time (`test_engine` E3).

The conditions are unanticipated: agents learn in quarter t that the variable was hit, and the endogenized shocks enter the data the learners see. A condition therefore moves beliefs and shares in later quarters.

14.4.1 `conditions`

Default an empty struct array.

Values a struct array, one element per pair, with fields

- `variable`: a transition or measurement variable;
- `shock`: a shock of the model;
- `dates`: dates inside the horizon, or `[start, end]`;
- `values`: a scalar or one value per date;
- `type`: `'level'` (default) or `'delta'`;
- `reference`: `'self'` (default) or a scenario id.

Math the targets τ_t of (33): the value itself (`'level'`), or the reference path plus the value (`'delta'`).

Assumptions `'self'` is the same scenario with every condition removed and `compare_with` emptied, run first and not saved (an inversion history is reused); a scenario id is read from `results_dir/<id>/results.mat`, and run first when it has no results.

Interactions the option is replaced as a whole, not merged. A variable conditioned twice in a quarter, a date outside the horizon, an unknown variable or shock, or a values/dates mismatch are errors (`hetlearn:engine:conditions`). Under Ramsey policy, quarters whose endogenized shocks are all rule shocks keep only rule-only targets (Section 15.3). In density draws the conditions stay in force, so endogenized shocks offset draws on the same shock. The density fast path needs no conditions.

Code `+engine/conditions_prepare.m`, `+engine/step.m`, `+engine/core.m` (references).

Example

```
cfg.conditions = struct('variable', 'ms', 'shock', 'em', 'dates', [qq
    (2022,3), qq(2023,2)], ...
    'values', -0.25, 'type', 'delta', 'reference', 'self');
```

14.4.2 `engine.conditions_weighting`

Default `'std'`.

Values `'std'`, `'none'`.

Math the two minimum-norm cases of (33).

Assumptions `'std'`: the smallest shocks in standard-deviation units are the most plausible.

Interactions matters only with more endogenized shocks than targets (conditions, the inversion); `'std'` needs a positive `std_<shock>` for every endogenized shock.

Code `+engine/step.m` (`solve_shocks`).

Example

```
cfg.engine.conditions_weighting = 'none';
cfg.history.mode = 'inversion';
```

14.5 The one-period step

14.5.1 `engine.step`

Default `'fast'`.

Values `'fast'`, `'iris'`.

Math `'fast'`: quarter t from the state-space form (1) of the solved model, cached after every successful solve: $\xi_t = \mathbf{T}x^b_{t-1} + \mathbf{R}\varepsilon_t + \mathbf{K}$, $y_t = \mathbf{Z}x^b_t + \mathbf{H}\varepsilon_t + \mathbf{D}$, with x^b_{t-1} read from the databank (element `v{-k}` is `v` at $t-1-k$) and missing shocks set to 0. `'iris'`: `simulate(m, d, t:t, 'prependInput', true)`.

Assumptions none; the two agree to rounding (10^{-14} per quarter, about 10^{-12} over 16 quarters).

Interactions a stored backward system has priority over both. `'fast'` needs plain state names. Bit-identity tests pin `'iris'`. The fast step is 2–3 times faster without conditions and about 20 times with the observables exogenized (the inversion).

Code `+engine/step.m`, `+engine/solve_model.m`, `+engine/core.m`.

Example

```
cfg.engine.step = 'iris';
cfg.history.mode = 'inversion';
```

14.6 Identification and comparisons

14.6.1 `id, title, description`

Default `''`.

Values text. `id` must be set (`hetlearn:run:id`).

Math none.

Assumptions `id` names the results folder `results/<id>/`; scenario ids are function names, found by `compare_with` and `'delta'` references, so they must be unique on the path.

Interactions `hetlearn.evaluate` requires unique variant ids.

Code `run.m`, `+engine/save_results.m`.

Example

```
cfg.id = 'my_learn';
cfg.title = 'Switching on inflation';
```

14.6.2 `compare_with`

Default `{}`.

Values a cellstr of scenario ids.

Math `compare_<id>.csv`: this run minus that run, every observable, over the horizon, model units.

Assumptions the other run has the same horizon dates.

Interactions a target without results is run first (`feval(id)`); a target that cannot be run leaves its file unwritten with a warning (`hetlearn:save_results:compare`). `hetlearn.evaluate` empties it.

Code `+engine/save_results.m`, `+engine/load_result.m`.

Example

```
cfg.compare_with = {'a1_re'};
cfg.id = 'a1_learn_variant';
```

14.6.3 `report.variables`

Default `{}`.

Values a cellstr; `report` is a free-form subtree.

Math none. Reserved: converted to a cellstr and read by no toolbox code.

Assumptions none.

Interactions none.

Code `load_options.m`.

Example

```
cfg.report.variables = {'pinfobs', 'robs'};
cfg.report.note = 'free-form';
```

15 Monetary policy: rules and Ramsey

15.1 The policy section of a package

Ramsey policy needs `spec.policy` with four fields (`extend_model.m`, `policy_setup`):

- `instrument`: a transition variable (SW07: `r`);
- `rule_label`: the exact, unique label of the policy rule in `!transition_equations`;
- `targets`: a struct of target expressions (names, numbers, `+ - * / ^`, parentheses and lags `x{-k}`; at least one transition variable). SW07: `pinf = 4*pinf`, `ygap = y - yf`, `r = 4*r`, `dr = 4*(r - r{-1})`;
- `discount`: a number or an expression, in which substitutions written as `$name$` are expanded; SW07 uses `$cbetabar$*$cgamma$`.

The generator wraps the labeled rule:

```
!if ramsey
    min((1 - ramsey_disc_on)*(<discount>) + ramsey_disc_on*ramsey_disc) w_pinf*(4*pinf)
    ^2 + w_ygap*(y - yf)^2 + ...;
!else
    <the policy rule>;
!end
```

and declares `w_<k>`, `ramsey_disc_on` and `ramsey_disc` unconditionally. The reference file is never wrapped; the collapse test runs with `ramsey = false`. A label that is not found sets `meta.policy.ok = false` and is an error only for `policy.mode = 'ramsey'` (`hetlearn:policy:rule`). Without `spec.policy` there is no Ramsey mode and no loss accounting.

15.2 The planner's problem

With `policy.mode = 'ramsey'` the model is loaded again with `ramsey = true` and `Model.fromFile(..., 'optimal', {'type', policy.type})`, so IRIS derives the first-order conditions of

$$\min \; E_t \sum_{h\geq 0} \delta^h \sum_k w_k \left(\text{target}_{k,t+h}\right)^2 \tag{34}$$

subject to every other model equation, including the expectation block with the beliefs and shares in force, which the planner, like every agent, treats as permanent (Section 2.2). The Lagrange multipliers of the plan (IRIS `Mu_Eq<k>`) are new transition variables of the Ramsey model. Under `'commitment'` the planner honors past promises through their lags; under `'discretion'` it re-optimizes each quarter and takes the private sector's leads as given. The toolbox ships a corrected copy of one IRIS file, `toolbox/iris_patches/+model/Insertable.m`, which `setup_paths` puts first on the path, because the optimal-policy parser of IRIS 20230622 fails on MATLAB R2025a.

Timing. The history is always the rule model: the RE filter is run with the rule and the RE-moment initial beliefs come from the rule model (for anchored RE agents see Section 15.5.1). The plan starts at the origin: the multipliers are 0 at and before T_0 (`+policy/prepare.m`, `+policy/forecast_setup.m`), and their values are stored every quarter so their lags carry forward. With learning the planner re-optimizes each quarter with the current beliefs and shares and honors past multipliers.

15.3 Rule-only names

The generator finds the variables used only by the rule and their own laws of motion, and the shocks used only there (SW07: `ms`, `em`; `meta.policy.rule_vars`, `.rule_shocks`). Under Ramsey policy they move nothing else. Shocks and conditions on them still run, with one warning per run (`hetlearn:policy:ruleInputs`); in quarters whose endogenized shocks are all rule shocks, only conditions on rule-only variables are kept (a target on the instrument cannot be met through a rule shock once the planner sets the instrument).

15.4 Loss accounting

Every run of a package with `spec.policy`, rule and Ramsey alike, reports (`+policy/loss.m`)

$$\mathcal{L} = \sum_{h=1}^{N} \delta^h \sum_k w_k \left(\text{target}_{k,T_0+h}\right)^2, \qquad \mathcal{L}_k = w_k \sum_{h=1}^{N} \delta^h \left(\text{target}_{k,T_0+h}\right)^2, \tag{35}$$

with the weights of `policy.loss` and $N =$ `policy.loss_horizon` (default H). If $N > H$ the scenario is continued from `res.state` for $N - H$ quarters with zero shocks, no conditions and learning on (a failed continuation gives NaN and a warning). Rule and Ramsey runs are therefore compared on the same weights. `res.loss` holds the total, the contribution of each target, the target paths, the per-quarter loss and the discount factors; `loss.csv` has one row per target and a total row.

15.5 Options

15.5.1 `policy.mode`

Default `'rule'`.

Values `'rule'`, `'ramsey'`.

Math `'rule'`: the package's rule; `'ramsey'`: problem (34) from the origin.

Assumptions see Section 15.2.

Interactions `'ramsey'` needs `spec.policy` with a found label (`hetlearn:policy:noSpec`, `hetlearn:policy:rule`); it is not supported with `expectations.re_info = 'lagged'` (`hetlearn:options:unsupported`: the plan with predetermined RE expectations has not been derived) nor with the N-type options; `hetlearn.decompose` refuses Ramsey results. Anchored RE agents keep the Taylor-rule forecast, so mode `'rational'` with anchored agents is not an RE Ramsey plan; the RE benchmark is `re_variant = 'model_consistent'`.

Code `load_model.m`, `+policy/setup.m`, `+policy/prepare.m`, `+policy/forecast_setup.m`.

Example

```
cfg.policy.mode = 'ramsey';
cfg.policy.type = 'commitment';
```

15.5.2 `policy.type`

Default `'commitment'`.

Values `'commitment'`, `'discretion'`.

Math IRIS optimal policy of the given type.

Assumptions commitment: the planner honors past promises (multipliers); discretion: re-optimizes taking expectations as given.

Interactions `'ramsey'` only. Discretion is indeterminate in IRIS (`MULTIPLE_STABLE`) at some parameter points: SW07 under RE, and any learning variable at the RE point where the model is loaded; interior shares solve.

Code `load_model.m`.

Example

```
cfg.policy.mode = 'ramsey';
cfg.policy.type = 'discretion';
```

15.5.3 `policy.loss`

Default `struct('pinf', 1, 'ygap', 0.5, 'r', 0.1, 'dr', 0)`.

Values a struct of non-negative finite weights by target name of `spec.policy`.

Math w_k in (34) and (35).

Assumptions a quadratic loss in the targets.

Interactions replaced as a whole, not merged: a target left out gets weight 0; an unknown name is `hetlearn:policy:unknownTarget`. The weights are model parameters (`w_<k>`), so changing them needs no new file.

Code `+policy/setup.m`, `+policy/loss.m`.

Example

```
cfg.policy.mode = 'ramsey';
cfg.policy.loss = struct('pinf', 1, 'ygap', 0.25, 'r', 0.5);   % dr gets
    0
```

15.5.4 `policy.discount`

Default `[]`: `spec.policy.discount` evaluated at the parameters.

Values `[]` or a number in $(0, 1]$.

Math δ in (34) and (35); in the model file the discount is $(1 - \texttt{ramsey_disc_on}) \cdot \delta_{\text{spec}} + \texttt{ramsey_disc_on} \cdot \texttt{ramsey_disc}$, with $\texttt{ramsey_disc_on} = 1$ when the option is set.

Assumptions none.

Interactions used by the loss of rule runs too.

Code `+policy/setup.m`, `+policy/eval_expr.m`.

Example

```
cfg.policy.mode = 'ramsey';
cfg.policy.discount = 0.99;
```

15.5.5 `policy.loss_horizon`

Default `[]`: `forecast.horizon`.

Values `[]` or a positive integer.

Math N in (35).

Assumptions beyond H the scenario continues with zero shocks and no conditions, with learning on.

Interactions the continuation uses the forecast model and the end state; it is reported in `res.loss.continuation`.

Code `+policy/loss.m`.

Example

```
cfg.forecast.horizon = 16;
cfg.policy.loss_horizon = 80;
```

16 Density forecasts

With `density.enabled = true`, `hetlearn.run` runs the history once and then repeats the forecast stage for each of N shock draws from the same origin state (`+density/run.m`).

Draws (`+density/draws.m`). For the shocks s of `density.shocks` and the quarters of `density.range` inside the horizon, draw j adds $E_{t,s}^{(j)}$ to the scenario's own shock:

$$\varepsilon_{s,t}^{(j)} = \varepsilon_{s,t}^{\text{scen}} + E_{t,s}^{(j)}, \qquad E_{t,s}^{(j)} = \begin{cases} \sigma_s \, z_{t,s}^{(j)}, \;\; z \sim N(0,1) \text{ i.i.d.} & \texttt{'normal'} \\ \left(p_{\iota_t^{(j)}} - \kappa \, \bar{p}\right)_s, \;\; \iota_t^{(j)} \sim U\{1, \ldots, P\} & \texttt{'bootstrap'} \end{cases} \tag{36}$$

where σ_s is the model's `std_<shock>` (`'calibrated'`) or the sample standard deviation (normalized by the number of values minus one) of the RE-smoothed shock over `density.sigma_range` (`'smoothed'`), and $p_1, \ldots, p_P$ are the vectors of RE-smoothed values of all drawn shocks at the dates of `sigma_range` where all are finite, with mean $\bar{p}$; $\kappa = 1$ with `density.demean` and 0 otherwise. The bootstrap draws whole dates, jointly across shocks, with replacement. Random numbers come from `RandStream('mt19937ar', 'Seed', seed)`, draw by draw, so the first k draws do not depend on N and the global stream is not touched. Draws are unanticipated, like every shock.

Paths. The *fast path* (`+density/fast_path.m`) applies when `density.fast_path` is true, the mode is `'rational'` and there are no conditions: at $n_1 = 1$ the solution does not depend on the beliefs or shares except through `x_exp1_BR`, which feeds nothing, so one `simulate(m, d, horizon, 'anticipate', false)` gives every other variable; the learning diagnostics are then replayed on the simulated path. It equals the general path to rounding (`test_density` D4). The *general path* calls `+engine/forecast.m` per draw, reusing the point forecast's first-quarter solve, so beliefs and shares respond to the drawn shocks. Conditions keep their targets in every draw.

Exclusions. A draw with a non-finite value, or any absolute value above 10^6, in a summarized series is `'explosive'`; a draw whose forecast stage stops (for example `hetlearn:engine:almUnstable` or `noSolution`) is `'error'`. Both are left out of the summaries and counted.

Summaries. For every observable, the learners' model share `n2_model_<x>` $= 1 - n_{1,t}^m$ and the belief parameters of each learning variable's PLM equation: the quantiles, the mean, the standard deviation and the point forecast (the run without draws), in model units and, for observables, in report units. The quantile of probability p (`+density/quantile.m`, the definition of MATLAB's Statistics Toolbox) places the sorted values $x_{(1)} \le \cdots \le x_{(N)}$ at probabilities $(i - 0.5)/N$ and interpolates linearly:

$$q(p) = (1 - w)\, x_{(\lfloor \pi \rfloor)} + w\, x_{(\lfloor \pi \rfloor + 1)}, \qquad \pi = \min\big(\max(Np + 0.5,\, 1),\, N\big), \quad w = \pi - \lfloor \pi \rfloor. \tag{37}$$

Outputs. `res.density` (in memory, not in `results.mat`), `density.mat` (variable `density`), `density_quantiles.csv` (variable, date, quantile, `value_model`, `value_report`), `density_events.csv` (one row per draw: draw, status, the counts of `projection_plm`, `projection_alm`, `projection_alm_shares`, `explosive_events`, `rls_missing`, `shares_nan`, `gain_bound`, then `max_abs` and the error message). A run without density deletes the density files of an earlier run of the same id.

16.1 Options

16.1.1 `density.enabled`

Default `false`.

Values true, false.

Math switches the draws (36) on.

Assumptions the history is computed once; only the forecast stage is repeated.

Interactions needs `density.shocks` (`hetlearn:options:missing`); `hetlearn.evaluate` switches it off.

Code: `run.m`, `+density/run.m`.

Example:

```
cfg.density.enabled = true;
cfg.density.shocks = {'epinf', 'ew'};
```

16.1.2 `density.shocks`

Default: `{}`.

Values: a cellstr of shocks of the model (`hetlearn:density:shocks` otherwise).

Math: the shocks s in (36); the others keep the scenario's values.

Assumptions: uncertainty comes from these shocks only.

Interactions: required when enabled; a shock that is also endogenized by a condition is offset by the condition.

Code: `+density/draws.m`.

Example:

```
cfg.density.enabled = true;
cfg.density.shocks = {'eb', 'eg', 'eqs'};
```

16.1.3 `density.sigma`

Default: `'calibrated'`.

Values: `'calibrated'`, `'smoothed'`.

Math: σ_s in (36): `std_<shock>` of the model, or the sample standard deviation of the RE-smoothed shock over `sigma_range`.

Assumptions: normal, independent shocks.

Interactions: `density.method = 'normal'` only; `'smoothed'` needs at least two smoothed values (`hetlearn:density:sigma`).

Code: `+density/draws.m`.

Example:

```
cfg.density.sigma = 'smoothed';
cfg.density.sigma_range = [qq(1966,1), qq(2019,4)];
```

16.1.4 `density.sigma_range`

Default: `[]`: the filter range.

Values: a date vector or `[start, end]`.

Math: the sample of smoothed shocks for `'smoothed'` and the bootstrap pool.

Assumptions: the shocks of this sample are representative of the future.

Interactions: `'smoothed'` and `'bootstrap'`; the sample means of the drawn shocks over it are reported in `draw_info.sample_mean`.

Code: `+density/draws.m`.

Example

```
cfg.density.method = 'bootstrap';
cfg.density.sigma_range = [qq(1985,1), qq(2019,4)];
```

16.1.5 `density.method`

Default `'normal'`.

Values `'normal'`, `'bootstrap'`.

Math the two cases of (36).

Assumptions `'bootstrap'` keeps the joint distribution of the shocks of one date (their contemporaneous correlation and fat tails), not their serial dependence.

Interactions `'bootstrap'` ignores `density.sigma`; it needs at least two pool dates (`hetlearn:density:bootstrap`).

Code `+density/draws.m`.

Example

```
cfg.density.method = 'bootstrap';
cfg.density.demean = true;
```

16.1.6 `density.demean`

Default `true`.

Values true, false.

Math subtracts $\bar{p}$ from the pool in (36).

Assumptions the smoothed shocks have non-zero sample means; demeaning centers the density on the scenario's own shocks.

Interactions `'bootstrap'` only.

Code `+density/draws.m`.

Example

```
cfg.density.method = 'bootstrap';
cfg.density.demean = false;
```

16.1.7 `density.ndraws`

Default 200.

Values a positive integer.

Math N.

Assumptions none.

Interactions the general path costs one forecast stage per draw (SW07: about 0.7 s per draw over 16 quarters; the fast path about 0.07 s).

Code `+density/draws.m`, `+density/run.m`.

Example

```
cfg.density.ndraws = 500;
cfg.density.seed = 1;
```

16.1.8 `density.seed`

Default `0`.

Values a non-negative integer.

Math the seed of the run's own `RandStream`.

Assumptions none.

Interactions the same seed gives the same draws in RE and learning runs, so their densities differ only through the model.

Code Example `+density/draws.m`.

```
cfg.density.seed = 42;
```

16.1.9 `density.quantiles`

Default `[0.05 0.16 0.5 0.84 0.95]`.

Values a vector of numbers in $(0, 1)$.

Math the probabilities p of (37).

Assumptions none.

Interactions none.

Code Example `+density/quantile.m`.

```
cfg.density.quantiles = [0.1 0.25 0.5 0.75 0.9];
```

16.1.10 `density.range`

Default `[]`: the horizon.

Values a date vector or `[start, end]`; dates outside the horizon are ignored (none inside is `hetlearn:density:range`).

Math the quarters t in (36).

Assumptions uncertainty only in these quarters.

Interactions none.

Code Example `+density/draws.m`.

```
cfg.density.range = [qq(2022,3), qq(2023,2)];
```

16.1.11 `density.keep_draws`

Default `false`.

Values true, false.

Math none: stores every draw (`density.draws.series.<name>`, $H \times N$, and the drawn shocks) in `density.mat`.

Assumptions none.

Interactions file size grows with N.

Code `+density/run.m`.

Example

```
cfg.density.keep_draws = true;
cfg.density.ndraws = 100;
```

16.1.12 `density.fast_path`

Default `true`.

Values true, false.

Math one linear simulation per draw instead of the quarter-by-quarter loop, when the mode is `'rational'` and there are no conditions.

Assumptions at $n_1 = 1$ beliefs and shares do not affect the path.

Interactions false forces the general path (used to test their equality).

Code `+density/fast_path.m`.

Example

```
cfg.expectations.mode = 'rational';
cfg.density.fast_path = false;
```

17 Historical shock decompositions

`D = hetlearn.decompose(res, vars, range, opts)` (`decompose.m`) decomposes the deviation of each variable in `vars` (default `{'pinfobs', 'dy'}`, SW07 names) from its RE steady state into one component per shock (or group), initial conditions and an intercept channel. `res` is a result struct or a saved scenario id; `range` the reporting dates (default: every date from the first one with initial conditions in `res.paths` to the end of the horizon).

The RE steady state comes from the RE model's state space: $x^b_{\text{ss}} = (I - \mathbf{T}_b)^{-1}\mathbf{K}_b$, $\xi_{\text{ss}} = \mathbf{T}x^b_{\text{ss}} + \mathbf{K}$, $y_{\text{ss}} = \mathbf{Z}x^b_{\text{ss}} + \mathbf{D}$.

Method `'iris'`. IRIS contributions of the RE model, `simulate(m, res.paths, range, 'anticipate', false, 'contributions', true)`, from the first date with initial conditions; `init` is the `Init+Const+Trends` column minus the steady state; `beliefs_constants` is the rest (the `Nonlinear` column and rounding; 0 for a linear RE model).

Method `'tv'`. A time-varying linear decomposition with the model in force in each quarter: the RE model ($n_1 = 1$, zero beliefs) before the learning model applies (the RE-smoothed history), the parameter values recorded by the inversion (`res.inversion.params`) over an inversion history, and over the horizon the beliefs ϕ_{t-1} and model shares $n^m_{1,t}$ from `res.learning` (the beliefs in force after the projection facility, reverted ones included). The model is re-solved when the parameters change. With $\mathbf{T}_t, \mathbf{R}_t, \mathbf{Z}_t, \mathbf{H}_t$ its state-space matrices (or its backward system),

$$\xi^c_t = \mathbf{T}_t x^{b,c}_{t-1} + \mathbf{R}_t \varepsilon^c_t, \qquad y^c_t = \mathbf{Z}_t x^{b,c}_t + \mathbf{H}_t \varepsilon^c_t, \tag{38}$$

without constants, for $c =$ each shock ($\varepsilon^c =$ that shock only, zero start) and $c =$ `init` (no shocks, start = the deviation of the initial state from the steady state). Then

$$\texttt{beliefs_constants}_t = (\xi_t - \xi_{\text{ss}}) - \sum_c \xi^c_t, \tag{39}$$

the time-varying intercepts (perceived means) and their propagation. The decomposition is additive by construction. On the RE model `'tv'` equals `'iris'` (toy 10^{-13}). The matrices with their constants rebuild `res.paths` quarter by quarter (`D.check.recon_maxerr`; `x_exp1_BR` is left out in RE-model quarters, where it holds the history-stage forecasts).

Reading. Slope beliefs and shares act inside every shock component, through each quarter's $\mathbf{T}_t$ and $\mathbf{R}_t$, so they are not a component of their own; `beliefs_constants` is the intercept channel only.

Output. `D.values.<v>` holds one column per component (dates × components, model units) and `D.labels` their labels; `D.check` holds the additivity error and, under `'tv'`, the reconstruction error `recon_maxerr`. `decomposition.csv` has the same values in long form; `docs/USER_GUIDE.md`, Section 6.1, lists its columns.

Restrictions. Ramsey results are refused (`hetlearn:decompose:ramsey`), and so are results of `history.mode = 'kalman'` (`hetlearn:options:unsupported`): the filtered history is not a path of the model with the filtered shocks, because the state covariance never collapses.

17.1 Call options

17.1.1 Call options of `hetlearn.decompose`

Default `groups = struct()`, `method = 'auto'`, `model = []` (loaded from `res.cfg`), `save = true`, `results_dir = results/`, `file = ''` (`<results_dir>/<id>/decomposition.csv`), `quiet = false`.

Values `groups`: a struct of shock lists; `method`: `'auto'`, `'iris'`, `'tv'`; `model`: `struct('spec', spec, 'm', m, 'info', info)`.

Math `'auto'` uses `'iris'` for mode `'rational'` and `'tv'` otherwise; a group's component is the sum of its members'.

Assumptions see above.

Interactions a shock in two groups or an unknown shock is `hetlearn:decompose:groups`; an unknown option is `hetlearn:decompose:option`. A learning run with the RE-smoothed history is decomposed with the RE model over the history (it is the RE path) and with the learning model over the horizon.

Code Example `decompose.m`.

```
G = struct('demand', {{'eb', 'eg', 'eqs'}}, 'monetary', {{'em'}});
D = hetlearn.decompose('a6_inv_learn', {'pinfobs'}, [qq(2019,1), qq
    (2026,2)], struct('groups', G));
```

18 Recursive forecast evaluation

`E = hetlearn.evaluate(name, variants, origins, opts)` (`evaluate.m`) runs every variant (a scenario function name or a cfg struct) at every origin and scores the forecasts of the observables.

Runs. Each variant's cfg gets the origin, `forecast.horizon = opts.horizon`, zero future shocks, no conditions, density off, `compare_with = {}` and `history.realtime = opts.realtime`; only the expectations side of the scenario matters. Each variant's model is loaded once. With `opts.realtime` the RE filter of each origin is run on data up to that origin, once per origin and filter group, and shared by the variants of the group (the group key holds the PLM spec and `model_params`; lagged variants add `re_info`, the learning set and `re_variant`). An `'inversion'` variant gets one inversion to the last origin on the full-sample filter, keeping its state at every origin, and each forecast starts from the state at its origin (the inversion is one-sided). The parameters stay at their full-sample values. A failing run is recorded in `settings.failures` and its errors are missing.

Errors and metrics. For variant v, origin i, horizon h and observable k, in report units (scale c_k),

$$e_{v,i,h,k} = c_k\big(y_{k,T_i+h} - \hat{y}_{v,k,T_i+h|T_i}\big), \tag{40}$$

with the actuals from the package database (the current vintage); targets without data are skipped. By sample (`full` and each subsample of target dates), variant, variable and h: n, $\mathrm{RMSE} = \sqrt{\overline{e^2}}$, $\mathrm{ME} = \bar{e}$, $\mathrm{MAE} = \overline{|e|}$ over the variant's errors; the relative RMSE $\mathrm{RMSE}_v/\mathrm{RMSE}_b$ against the benchmark b on the targets both have.

Diebold–Mariano test (Diebold and Mariano, 1995). On the common targets, in origin order, $d_i = e_{v,i}^2 - e_{b,i}^2$ (a negative mean favors the variant), $u = d - \bar{d}$, $\hat{\gamma}_k = \frac{1}{n}\sum_{i>k} u_i u_{i-k}$, $L = \min(h-1, n-1)$, and

$$\widehat{\mathrm{LRV}} = \begin{cases} \hat{\gamma}_0 + 2\sum_{k=1}^{L} \hat{\gamma}_k & \texttt{'uniform'} \\ \hat{\gamma}_0 + 2\sum_{k=1}^{L} (1 - \frac{k}{L+1})\hat{\gamma}_k & \texttt{'bartlett'} \end{cases} \qquad \mathrm{DM} = \frac{\bar{d}}{\sqrt{\widehat{\mathrm{LRV}}/n}}. \tag{41}$$

The uniform kernel is the rectangular window for which Harvey et al. (1997) (HLN) derived their correction; where its variance is not positive the cell falls back to Bartlett (Newey and West, 1987) (kernel column `bartlett_fallback`, counted in `settings.dm_fallbacks`). With the HLN correction the statistic is multiplied by $\sqrt{(n+1-2h+h(h-1)/n)/n}$ and the two-sided p-value comes from Student's t with $n-1$ degrees of freedom, $p = I_{\nu/(\nu+\mathrm{DM}^2)}(\nu/2, 1/2)$; without it $p = \mathrm{erfc}(|\mathrm{DM}|/\sqrt{2})$. DM is NaN when $n < 2$ or the variance is not positive.

Outputs (`results/eval_<name>/`): the errors in `errors_long.csv`, the tables `metrics.csv`, `relative_rmse.csv` and `dm.csv`, and `settings.mat`; `docs/USER_GUIDE.md`, Section 6.4, lists their columns. `E` also holds the forecasts and actuals as arrays. `hetlearn.evaluate_`

`tables(errors, opts)` rebuilds the three tables from a saved `errors_long.csv` (or an `eval_<name>` folder) without rerunning any forecast, for example with another kernel.

18.1 Call options

18.1.1 Call options of `hetlearn.evaluate`

Default `horizon = 8`, `benchmark = ''` (the first variant), `dm = true`, `dm_hln = true`, `dm_kernel = ''` (`'uniform'` with HLN, else `'bartlett'`), `subsamples = struct()`, `realtime = true`, `results_dir = results/`, `save = true`, `quiet = false`.

Values `subsamples`: a struct of `[start, end]` target-date ranges; `dm_kernel`: `''`, `'uniform'`, `'bartlett'`; `origins`: a date vector or `[start, end]`.

Math equations above.

Assumptions real-time information for the history and the starting state (with `realtime`); full-sample parameters; actuals of the current vintage.

Interactions all variants must use the same package (`hetlearn:evaluate:model`) and have unique ids; the benchmark must be a variant. Lagged and inversion variants get their own filters as described above. `evaluate_tables` takes the table options, with defaults from `settings.mat` next to the file (`docs/USER_GUIDE.md`, Section 7).

Code Example `evaluate.m`, `evaluate_tables.m`.

```
E = hetlearn.evaluate('my_eval', {'e01_re', 'e02_learn'}, [qq(2010,1), qq
    (2024,2)], ...
    struct('horizon', 8, 'subsamples', struct('post2008', [qq(2008,1), qq
    (2019,4)])));
```

19 Scenario files and caches

19.1 Scenario files

A scenario is a function that returns a partial option tree. Inheritance is a function call: a file starts from its parent and sets what differs.

```
function cfg = my_learn()
cfg = a1_re();                          % the RE parent: model, origin, horizon, learning
    set
cfg.id = 'my_learn';                    % results/my_learn/
cfg.expectations.mode = 'switching';
cfg.compare_with = {'a1_re'};
end
```

Ids are function names; `compare_with`, `'delta'` references and the runner find scenarios by name, so names must be unique on the path. Each result stores its merged options (`res.cfg`); `hetlearn.load_model` re-merges a saved tree, so it runs on later versions. Saved results never update themselves: after an edit, rerun the scenario and its descendants.

19.2 Caches

Generated files and RE filter caches live in `<package>/generated/` (Sections 4.5 and 13.1). Every cache key includes what the cached object depends on (model text, parameters, toggles, data, range), so a stale cache is never used; deleting the folder only forces a rebuild.

20 Outputs and the result struct

`res = hetlearn.run(cfg)` returns the result struct and, unless `'save', false`, writes `results/<id>/` (`+engine/assemble.m`, `+engine/save_results.m`). `docs/USER_GUIDE.md`, Section 6, describes the same files for the project's scenarios.

20.1 The result struct

Field	Content
`cfg`	the merged option tree
`paths`	databank over [filter start, $T_0 + H$] of every transition and measurement variable and shock: the history to the origin (RE-smoothed, backed out over the inversion, or filtered under the Kalman history), the scenario after it. `x_exp1_BR` holds the learners' forecasts over the record dates and NaN before
`learning`	struct by PLM target (Section 20.2)
`events`	table: `date`, `type`, `variable`, `detail`, `stage` (Section 12.5)
`state`	the learning state at the end of the horizon: `phi`, `R`, `zetaRE`, `zetaBR`, `n1`, `fRE`, `fBR`, `fe_hist`, `phi_hist`, `gain`, `prev`, `n1m_last`, `n1m_keep`
`dates`	`origin`, `horizon`, `init_date`, `record` ($t_0, \ldots, T_0 + H$, or $T_0, \ldots$ without history learning), `full`
`meta`	model, folder, learning variables, name lists, extended file and its MD5, collapse-test results, RE filter hash, fit error, end date and real-time flag, IRIS version, `history_mode`, `engine_step`, `projection_facility`, `history_runtime_s`, `history_alm_check_s`, `history_alm_n_check`, `history_alm_n_solve`, `forecast_n_solve`, `started`, `finished`, `runtime_s`
`loss`	the policy loss (Section 15.4); empty without `spec.policy`
`history_mode`	`history.mode`
`inversion`	inversion diagnostics (Section 13.3)
`kalman`	Kalman history output (Section 13.4)
`density`	density output, in memory only (Section 16)

20.2 Learning fields

`res.learning.<target>` holds IRIS Series over the record dates for every PLM equation, auxiliary ones included:

- one series per belief parameter (`alpha_pinf`, `beta_pinf`, `beta_pinf__ygap_l1`, ...): ϕ_t, the beliefs at the end of t, in force in $t + 1$ (after the facility, reverted ones included);
- `R_<param>`: the diagonal of R_t;
- `gain`: the gain used in t (NaN where there was no update); `gain_R` under KOZ;
- `isAux`, `params` (the parameter names).

For learning variables also: `n1`, `n2` (discrete-choice shares at the end of t), `n1_model` (the RE share in the model in t; NaN over the RE-smoothed history), `zetaRE`, `zetaBR`, `FE_RE`, `FE_BR` (the variable's own squared errors; under universal shares the group sums drive the shares) and `fRE`, `fBR` (forecasts of x_{t+1} made in t). Under switching, `n1_model`(t) = `n1`$(t-1)$ unless the facility kept the shares (`test_engine` E11 checks the timing from the paths alone).

20.3 Share names

Name	Where	Meaning
`shares.init_n2`, `shares.fixed_n2`	options	learners' share
`n1_<x>`, `n2_<x>`	model parameters	RE and learners' shares of x's expectation in the model
`n1`, `n2`	`res.learning.<x>`	discrete-choice shares at the end of t
`n1_model`	`res.learning.<x>`	RE share in the model in t
`n2_model_<x>`	density outputs	1−`n1_model`

20.4 Files in `results/<id>/`

File	Content
`results.mat`	`res` (without `density`)
`observables.csv`	date and every observable, [filter start, $T_0 + H$], model units
`paths.csv`	date and every transition variable over the horizon
`learning.csv`	long: date, variable (PLM target), field, value
`events.csv`	date, type, variable, detail, stage
`compare_<id>.csv`	this run minus run `<id>`, observables, horizon
`loss.csv`	target, expression, weight, contribution, share, discount, horizon, continuation, mode, type; a total row
`density.mat`, `density_quantiles.csv`, `density_events.csv`	Section 16
`decomposition.csv`	after `hetlearn.decompose`
`kalman_errors.csv`	Kalman history (a stale file is deleted otherwise)

Dates are written as text (`2022Q3`); numbers with 17 significant digits. Observables and learning series are in model units; `value_report` columns multiply by `spec.report.scale`.

21 Validation and the test suite

21.1 Checks inside every run

Check	What it compares	Tolerance
Collapse test	extended model at $n_1 = 1$, zero beliefs, against the original (Section 4.6)	10^{-10}
PLM consistency	the run's BR forecast text against the generated file	exact text

Check	What it compares	Tolerance
Anchored coefficients	state-space coefficients against unit-perturbation simulations	10^{-9} relative
Lagged anchored model	anchored against model-consistent RE(t-1) model at $n_1 = 1$	10^{-10}
RE filter fit	observables rebuilt through the measurement equations	10^{-8}
Timing check	model's `x_exp1_BR` against the PLM formula with ϕ_{t-1}	10^{-9}
Conditions	every target met after the shock solve	10^{-9}
Kalman	positive definite F_t; stationary first ALM under `'stochastic'`	roots below $1 - 10^{-10}$

21.2 The test suite

`tests/run_tests.m` runs the test suites of the toolbox and prints one PASS/FAIL line per check; `run_tests({'test_options', 'test_rls'})` runs a subset. The suites check the equations of this appendix against hand-coded loops, IRIS references and stored results in `tests/reference/`, on the toy package `tests/toy_nk/` (a three-equation hybrid New Keynesian model, Özden, 2025, eq. 1) and on SW07 test scenarios (`tests/scenarios/tsw07_*.m`).

21.3 Known limitations

- Shocks and conditions are unanticipated (Section 2.2).
- Nonlinear models and log variables: Section 3.1.
- Ramsey policy: Sections 15.5.1 and 15.5.2.
- `history.realtime` acts on the RE-smoother history only (Section 13.2.3).
- The evaluation uses the current data vintage and full-sample parameters (Section 18).
- With zero future shocks the shares drift toward 1/2 (Section 11.1).

A Option index

Every option of `hetlearn.defaults()` within the scope of this appendix, with its default and the entry that documents it. Defaults in brackets are resolved later (`[]` or `''`).

Option	Default	Section	Page
`id`, `title`, `description`	`''`	14.6.1	49
`model`	`''`	3.3.1	11
`model_params`	`''` (= `spec.params`)	3.3.2	11
`forecast.origin`	`[]` (required)	14.2.1	45
`forecast.horizon`	`16`	14.2.2	45
`expectations.mode`	`'rational'`	5.1.1	15
`expectations.variables`	`{}`	5.1.2	16
`expectations.re_variant`	`'anchored'`	6.4.1	18
`expectations.re_info`	`'current'`	7.2.1	20
`beliefs.update`	`'rls'`	9.1.1	24
`beliefs.info`	`'lagged'`	7.1.1	19
`beliefs.plm.default`	`'ar1'`	8.2.1	21
`beliefs.plm.by_var`	`struct()`	8.2.2	22

Option	Default	Section	Page
beliefs.plm.derived	struct()	8.2.3	23
beliefs.plm.var	no VAR (lags 1, constant)	8.2.4	23
beliefs.init.method	're_moments'	10.1.1	30
beliefs.init.date	[] (filter start + 4)	10.1.2	31
beliefs.init.values	struct()	10.1.3	32
beliefs.init.training_range	[]	10.1.4	32
gain.rule	'constant'	9.2.1	25
gain.value	0.0585	9.2.2	25
gain.by_var	struct()	9.2.3	26
gain.koz.mu	0.001	9.4.1	27
gain.koz.gamma0	[] (= gain.value)	9.4.2	28
gain.koz.Rg0	0	9.4.3	28
gain.koz.bounds	[0.001 0.25]	9.4.4	28
gain.koz.error	'rls'	9.4.5	29
gain.koz.error_scale	1	9.4.6	29
gain.koz.structure	'common'	9.4.7	30
gain.koz.master	'' (first learning variable)	9.4.8	30
gain.decreasing.t0	0	9.3.1	26
shares.structure	'per_variable'	11.2.6	35
shares.master	{}	11.2.7	35
shares.init_n2	0.5	11.2.4	34
shares.fixed_n2	[]	11.2.5	35
switching.rule	'discrete_choice'	11.2.1	33
switching.intensity	0.51	11.2.2	34
switching.memory	0.6	11.2.3	34
projection.facility	'plm_alm'	12.2.1	36
history.learn	true	13.2.4	41
history.mode	're_smoother'	13.2.1	40
history.re_states	'smoothed'	13.2.2	40
history.realtime	false	13.2.3	41
history.inversion.start	[] (= init date)	13.3.1	42
history.inversion.shocks	{} (transition shocks)	13.3.2	42
history.kalman.init	'stochastic'	13.4.1	44
shocks.source	'none'	14.3.1	46
shocks.range	[] (horizon)	14.3.2	46
shocks.exclude	{}	14.3.3	46
shocks.add	{}	14.3.4	47
conditions	empty struct array	14.4.1	48
density.enabled	false	16.1.1	54
density.shocks	{}	16.1.2	55
density.sigma	'calibrated'	16.1.3	55
density.sigma_range	[] (filter range)	16.1.4	55
density.method	'normal'	16.1.5	56
density.demean	true	16.1.6	56
density.ndraws	200	16.1.7	56
density.seed	0	16.1.8	57
density.quantiles	[0.05 0.16 0.5 0.84 0.95]	16.1.9	57
density.range	[] (horizon)	16.1.10	57
density.keep_draws	false	16.1.11	58

Option	Default	Section	Page
`density.fast_path`	`true`	16.1.12	58
`engine.step`	`'fast'`	14.5.1	49
`engine.conditions_weighting`	`'std'`	14.4.2	48
`policy.mode`	`'rule'`	15.5.1	51
`policy.type`	`'commitment'`	15.5.2	52
`policy.loss`	`pinf` 1, `ygap` 0.5, `r` 0.1, `dr` 0	15.5.3	52
`policy.discount`	`[]` (= spec)	15.5.4	53
`policy.loss_horizon`	`[]` (= horizon)	15.5.5	53
`compare_with`	`{}`	14.6.2	49
`report.variables`	`{}` (reserved)	14.6.3	50

Not covered here (bounded-rationality extensions, off by default): `expectations.types`, `expectations.transforms`, `shares.init`, `shares.fixed`. Call options of `hetlearn.decompose` and `hetlearn.evaluate` are in Sections 17.1.1 and 18.1.1; those of `hetlearn.extend_model` in Section 4.7.

References

William A. Brock and Cars H. Hommes. A rational route to randomness. *Econometrica*, 65(5): 1059–1095, 1997.

Francis X. Diebold and Roberto S. Mariano. Comparing predictive accuracy. *Journal of Business and Economic Statistics*, 13(3):253–263, 1995.

James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, Oxford, second edition, 2012.

George W. Evans and Seppo Honkapohja. *Learning and Expectations in Macroeconomics*. Princeton University Press, Princeton, NJ, 2001.

David Harvey, Stephen Leybourne, and Paul Newbold. Testing the equality of prediction mean squared errors. *International Journal of Forecasting*, 13(2):281–291, 1997.

IRIS Solutions Team. IRIS macroeconomic modeling toolbox. MATLAB toolbox, release 20230622, 2023. https://github.com/IRIS-Solutions-Team/IRIS-Toolbox.

Olena Kostyshyna. Application of an adaptive step-size algorithm in models of hyperinflation. *Macroeconomic Dynamics*, 16:355–375, 2012.

Olena Kostyshyna, Tolga Özden, and Yang Zhang. Endogenous credibility and wage-price spirals. Staff Working Paper 2024-14, Bank of Canada, 2024.

David M. Kreps. Anticipated utility and dynamic choice. In Donald P. Jacobs, Ehud Kalai, and Morton I. Kamien, editors, *Frontiers of Research in Economic Theory: The Nancy L. Schwartz Memorial Lectures, 1983–1997*, pages 242–274. Cambridge University Press, Cambridge, 1998.

Harold J. Kushner and G. George Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer, New York, second edition, 2003.

Albert Marcet and Thomas J. Sargent. Convergence of least squares learning mechanisms in self-referential linear stochastic models. *Journal of Economic Theory*, 48(2):337–368, 1989.

Whitney K. Newey and Kenneth D. West. A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica*, 55(3):703–708, 1987.

Tolga Özden. Heterogeneous expectations and the business cycle. *International Journal of Central Banking*, 21(1), 2025.

Frank Smets and Rafael Wouters. Shocks and frictions in US business cycles: A Bayesian DSGE approach. *American Economic Review*, 97(3):586–606, 2007.