RISE Documentation

Release 1.0.1

Junior Maih

November 20, 2014

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 RISE at a Glance

1.1.1 What is RISE?

RISE is the acronym for **R**ationality **I**n **S**witching **E**nvironments.

It is an object-oriented Matlab toolbox primarily designed for solving and estimating nonlinear dynamic stochastic general equilibirium (**DSGE**) or more generally Rational Expectations(**RE**) models with **switching parameters**.

Leading references in the field include various papers by Roger Farmer, Dan Waggoner and Tao Zha and Eric Leeper among others.

RISE uses perturbation to approximate the nonlinear Markov Switching Rational Expectations (**MSRE**) model and solves it using efficient algorithms.

RISE also implements special cases of the general Switching MSRE model. This includes

- **VAR**s with and without switching parameters
- **SVAR**s with and without switching paramters
- **Time-varying parameter VAR**s
- etc.

1.1.2 Motivation for RISE development

- The world is not constant, it is switching

1.2 Capabilities of RISE

1.2.1 DSGE modeling

- constant parameters
- **switching parameters**
    - exogenous switching
    - endogenous switching
- **optimal policy (with and without switching)**

    - discretion
    - commitment
    - loose commitment
    - optimized simple rules
- Deterministic simulation
- Stochastic simulation
- higher-order perturbations

1.2.2 VAR modeling

- **constant parameters**
    - zero restrictions
    - sign restrictions
    - restrictions on lag structure
    - linear restrictions
- **switching parameters**
    - linear restrictions

1.2.3 SVAR modeling

- constant parameters
- **switching parameters**
    - linear restrictions

1.2.4 Time-Varying parameter VAR modeling

Under implementation

1.2.5 Smooth transition VAR modeling

Not yet implemented

1.2.6 Forecasting and Conditional Forecasting

1.2.7 Global sensitivity analysis

- Monte carlo filtering
- High dimensional model representation

1.2.8 Maximum Likelihood and Bayesian Estimation

- linear restrictions
- nonlinear restrictions

1.2.9 Time series

1.2.10 Reporting

1.3 How RISE works

1.3.1 Object orientation

1.3.2 Basic principles

- you can pass different options at any time

1.4 Background and mathematical formulations

1.5 Using this documentation

1.5.1 how to find help

1.5.2 Road map

1.6 Citing RISE in your research

1.7 License and Legal Mumbo-Jumbo

CHAPTER

TWO

GETTING STARTED WITH RISE

2.1 Installation guide

2.1.1 Software requirements

I order to use RISE, the following software will need to be installed:

- Matlab version ? or higher
- MikTex (Windows users) MacTex (mac users)

2.1.2 How to obtain RISE

There are (at least) two ways to acquire RISE:

The zip file option

1. Go online to https://github.com/jmaih/RISE_toolbox
2. download the zip file and unzip it in some directory on your computer.

This option is not recommended but is convenient for people who are not allowed to install new software on their machines/laptop.

Github for the bleeding-edge installation (highly recommended)

1. Go to http://windows.github.com if you are a windows user or to http://mac.github.com if you are a mac user
2. Create an account online through the website and download the Github program
3. Sign in both online and on the github on your machine. It is obvious online, but on your machine, just go to Github>Preference>Account
4. Go online to https://github.com/jmaih/RISE_toolbox
5. Look for an icon with title 'Clone in Desktop' (or possibly clone in mac). There are options to locate where the repository will reside

The reason why this option is recommended is that you don't need to re-download the whole toolbox every time a marginal update is made. With one click and within seconds you can have the version of the toolbox on your computer updated.

The git option (never tested!!!)

The following has never been tested and so the syntax might be wrong:

```
git clone https://github.com/jmaih/RISE_toolbox.git
```

Testing your installation

More on this later...

2.1.3 Loading and starting RISE

1. Locate the RISE_toolbox directory and add its path to matlab in the command window as

   ```
   addpath('C:/Users/JMaih/GithubRepositories/RISE_toolbox')
   ```

2. You will need to adapt this path to conform with the location of the toolbox on your machine.
3. run rise_startup()

2.1.4 Updating RISE

New features are constantly added, efficiency is improved, users sometimes report bugs that are corrected. All this makes it necessary to update RISE every now and then in order to keep abreast of the latest changes and developments.

However, updating RISE depends on precisely how you installed it in the first place:

- If you downloaded a zip file, you will have to redownload a zip file even if the recent change was just an added comma.
- if instead you invested in opening a github account, with one click you will be able to update just the changes you don't have.
- with git, you would just execute the command

  ```
  git pull
  ```

2.2 Troubleshooting

2.3 RISE basics/basic principles

1. create an empty RISE object e.g.

   ```
   tao=rise.empty(0);
   ```

2. run methods(rise) or methods(tao) to see the functions/methods that can be applied to a RISE object
3. run those methods on r". e.g. "irf(r)", simulate(r)", solve(r)", etc. this will give you the default options of each method and tell you how you can modify the behavior of the method

2.4 Tutorial: A toy example

2.4.1 Foerster, Rubio-Ramirez, Waggoner and Zha (2014)

They consider the following model:

$$E_t \left[\begin{array}{c} 1-\beta \frac{\left(1-\frac{\kappa}{2}(\Pi_t-1)^2\right)Y_t}{\left(1-\frac{\kappa}{2}(\Pi_{t+1}-1)^2\right)Y_{t+1}} \frac{1}{e^{\mu_{t+1}}} \frac{R_t}{\Pi_{t+1}} \\ (1-\eta)+\eta\left(1-\frac{\kappa}{2}\left(\Pi_t-1\right)^2\right)Y_t+\beta\kappa\frac{\left(1-\frac{\kappa}{2}(\Pi_t-1)^2\right)}{\left(1-\frac{\kappa}{2}(\Pi_{t+1}-1)^2\right)}\left(\Pi_{t+1}-1\right)\Pi_{t+1}-\kappa\left(\Pi_t-1\right)\Pi_t \\ \left(\frac{R_{t-1}}{R_{ss}}\right)^{\rho}\Pi_t^{(1-\rho)\psi}\exp\left(\sigma\varepsilon_t\right)-\frac{R_t}{R_{ss}} \end{array} \right] = 0$$

$$with$$

$$\mu_{t+1} = \bar{\mu} + \sigma\hat{\mu}_{t+1}.$$

The first equation is an Euler equation, the second equation a Phillips curve and the third equation a nonlinear Taylor rule.

The switching parameters are μ and ψ.

2.4.2 The RISE code

The RISE code with parameterization is given by

```
endogenous PAI,Y,R

exogenous EPS_R

parameters a_tp_1_2, a_tp_2_1, betta, eta, kappa, mu, mu_bar, psi, rhor, sigr
parameters(a,2) mu, psi

model
        1-betta*(1-.5*kappa*(PAI-1)^2)*Y*R/((1-.5*kappa*(PAI(+1)-1)^2)*Y(+1)*exp(mu)*PAI(+1));

        1-eta+eta*(1-.5*kappa*(PAI-1)^2)*Y+betta*kappa*(1-.5*kappa*(PAI-1)^2)*(PAI(+1)-1)*PAI(+1)/(1-
        -kappa*(PAI-1)*PAI;

        (R(-1)/steady_state(R))^rhor*(PAI/steady_state(PAI))^((1-rhor)*psi)*exp(sigr*EPS_R)-R/steady_


steady_state_model(unique,imposed)
    PAI=1;
    Y=(eta-1)/eta;
    R=exp(mu_bar)/betta*PAI;


parameterization
        a_tp_1_2,1-.9;
        a_tp_2_1,1-.9;
        betta, .99;
        kappa, 161;
        eta, 10;
        rhor, .8;
        sigr, 0.0025;
        mu_bar,0.02;
        mu(a,1), 0.03;
```

```
        mu(a,2), 0.01;
        psi(a,1), 3.1;
        psi(a,2), 0.9;
```

2.4.3 Running the example

Assume this example is saved in a file named frwz_nk.rs . The to run this example in Matlab, we run the following commands:

```
frwz=rise('frwz_nk'); % load the model and its parameterization

frwz=solve(frwz); % Solving the model

print_solution(frwz) % print the solution
```

2.5 How to find help?

2.6 Where to go from here

CHAPTER

THREE

RISE CAPABILITIES

3.1 Overview

3.2 Markov switching DSGE modeling

3.3 Markov switching SVAR modeling

3.4 Markov switching VAR modeling

3.5 Smooth transition VAR modeling

3.6 Time-varying parameter modeling

3.7 Maximum Likelihood and Bayesian Estimation

3.8 Differentiation

3.8.1 numerical differentiation

3.8.2 Symbolic differentiation

3.8.3 Automatic/Algorithmic differentiation

3.9 Time series

3.10 Reporting

3.11 Derivative-free optimization

3.12 Global sensitivity analysis

3.12.1 Monte Carlo filtering

3.12.2 High dimensional model representation

CHAPTER

FOUR

THE MARKOV SWITCHING DSGE INTERFACE

4.1 The general framework

The general form of the models is:

$$E_t \sum_{r_{t+1}=1}^{h} \pi_{r_t,r_{t+1}} (I_t)\, \tilde{d}_{r_t} \left(b_{t+1} (r_{t+1}), b_t (r_t), b_{t-1}, \varepsilon_t, \theta_{r_{t+1}}\right) = 0$$

- The switching of the parameters is governed by Markov processes and can be endogenous.
- Agents can have information about future events

4.2 The model file

4.2.1 Conventions

4.2.2 Variable declarations

4.2.3 Expressions

- **parameters and variables**
  - inside the model
  - outside the model
- operators
- **functions**
  - built-in functions
  - external/user-defined functions

4.2.4 model declaration

- model equations

- endogenous transition probabilities
- auxiliary parameters/variables
- inequality restrictions

4.2.5 auxiliary variables

4.2.6 initial and terminal conditions

4.2.7 shocks on exogenous variables

4.2.8 other general declarations

4.3 steady state

- finding the steady state with the RISE nonlinear solver
- using a steady state file
- using the steady state model

4.4 getting information about the model

4.5 deterministic simulation

4.6 stochastic solution and simulation

- computing the stochastic solution
- typology and ordering of variables
- first-order approximation
- second-order approximation
- third-order approximation
- fourth-order approximation
- fifth-order approximation

4.7 Estimation

4.8 Forecasting and conditional forecasting

4.9 Optimal policy

- optimal simple rules

- Commitment, discretion and loose commitment

CHAPTER

FIVE

MARKOV SWITCHING DYNAMIC STOCHASTIC GENERAL EQUILIBRIUM MODELING

5.1 methods

- [check_derivatives](dsge/check_derivatives)
- [check_optimum](dsge/check_optimum)
- [compute_steady_state](dsge/compute_steady_state)
- [create_estimation_blocks](dsge/create_estimation_blocks)
- [create_state_list](dsge/create_state_list)
- [draw_parameter](dsge/draw_parameter)
- [dsge](dsge/dsge)
- [estimate](dsge/estimate)
- [filter](dsge/filter)
- [forecast](dsge/forecast)
- [forecast_real_time](dsge/forecast_real_time)
- [get](dsge/get)
- [historical_decomposition](dsge/historical_decomposition)
- [irf](dsge/irf)
- [is_stable_system](dsge/is_stable_system)
- [isnan](dsge/isnan)
- [load_parameters](dsge/load_parameters)
- [log_marginal_data_density](dsge/log_marginal_data_density)
- [log_posterior_kernel](dsge/log_posterior_kernel)
- [log_prior_density](dsge/log_prior_density)
- [monte_carlo_filtering](dsge/monte_carlo_filtering)
- [posterior_marginal_and_prior_densities](dsge/posterior_marginal_and_prior_densities)

- [posterior_simulator](dsge/posterior_simulator)
- [print_estimation_results](dsge/print_estimation_results)
- [print_solution](dsge/print_solution)
- [prior_plots](dsge/prior_plots)
- [report](dsge/report)
- [resid](dsge/resid)
- [set](dsge/set)
- [set_solution_to_companion](dsge/set_solution_to_companion)
- [simulate](dsge/simulate)
- [simulate_nonlinear](dsge/simulate_nonlinear)
- [simulation_diagnostics](dsge/simulation_diagnostics)
- [solve](dsge/solve)
- [solve_alternatives](dsge/solve_alternatives)
- [stoch_simul](dsge/stoch_simul)
- [theoretical_autocorrelations](dsge/theoretical_autocorrelations)
- [theoretical_autocovariances](dsge/theoretical_autocovariances)
- [variance_decomposition](dsge/variance_decomposition)

5.2 properties

- [definitions] -
- [equations] -
- [folders_paths] -
- [dsge_var] -
- [filename] -
- [legend] -
- [endogenous] -
- [exogenous] -
- [parameters] -
- [observables] -
- [markov_chains] -
- [options] -
- [estimation] -
- [solution] -
- [filtering] -

5.3 check_derivatives

check_derivatives - compares the derivatives and the solutions from various differentiation techniques

5.3.1 Syntax

```
check_derivatives(obj)
retcode=check_derivatives(obj)
```

5.3.2 Inputs

- **obj** [riseldsge]: model object or vectors of model objects

5.3.3 Outputs

- **retcode** [numeric]: 0 if no problem is encountered during the comparisons. Else the meaning of recode can be found by running decipher(retcode)

5.3.4 More About

- The derivatives computed are ‘automatic’, ‘symbolic’ or ‘numerical’
- The comparisons are done relative to automatic derivatives, which are assumed to be the most accurate.

5.3.5 Examples

See also:

5.4 check_optimum

H1 line

5.4.1 Syntax

5.4.2 Inputs

5.4.3 Outputs

5.4.4 More About

5.4.5 Examples

See also:

Help for dsge/check_optimum is inherited from superclass RISE_GENERIC

5.5 compute_steady_state

H1 line

5.5.1 Syntax

5.5.2 Inputs

5.5.3 Outputs

5.5.4 More About

5.5.5 Examples

See also:

5.6 create_estimation_blocks

H1 line

5.6.1 Syntax

5.6.2 Inputs

5.6.3 Outputs

5.6.4 More About

5.6.5 Examples

See also:

5.7 create_state_list

create_state_list creates the list of the state variables in the solution

5.7.1 Syntax

```
final_list=create_state_list(m)
final_list=create_state_list(m,orders)
```

5.7.2 Inputs

- **m** [dsge|rise] : model object
- **orders** [integer array|{1:m.options.solve_order}] : approximation orders
- **compact_form** [true|{false}] : if true, only unique combinations will be returned. Else, all combinations will be returned.

5.7.3 Outputs

- **final_list** [cellstr] : list of the state variables
- **kept** [vector] : location of kept state variables (computed only if compact_form is set to true)

5.7.4 More About

5.7.5 Examples

See also:

5.8 draw_parameter

H1 line

5.8.1 Syntax

5.8.2 Inputs

5.8.3 Outputs

5.8.4 More About

5.8.5 Examples

See also:

Help for dsge/draw_parameter is inherited from superclass RISE_GENERIC

5.9 dsge

~~ no help found

5.10 estimate

estimate - estimates the parameters of a RISE model

5.10.1 Syntax

```
obj=estimate(obj)
obj=estimate(obj,varargin)
```

5.10.2 Inputs

- **obj** [rise|dsge|rfvar|svar]: model object
- **varargin** additional optional inputs among which the most relevant for estimation are:
- **estim_parallel** [integer|{1}]: Number of starting values
- **estim_start_from_mode** [true|false|{[]}]: when empty, the user is prompted to answer the question as to whether to start estimation from a previously found mode or not. If true or false, no question is asked.
- **estim_start_date** [numeric|char|serial date]: date of the first observation to use in the dataset provided for estimation
- **estim_end_date** [numeric|char|serial date]: date of the last observation to use in the dataset provided for estimation
- **estim_max_trials** [integer|{500}]: When the initial value of the log-likelihood is too low, RISE uniformly draws from the prior support in search for a better starting point. It will try this for a maximum number of **estim_max_trials** times before squeaking with an error.

- **estim_start_vals** [{[]}|struct]: when not empty, the parameters whose names are fields of the structure will see their start values updated or overriden by the information in **estim_start_vals**. There is no need to provide values to update the start values for the estimated parameters.
- **estim_general_restrictions** [{[]}|function handle]: when not empty, the argument should be a function handle that takes as input a parameterized RISE object and returns a scalar or vector of numbers representing the strength of the violation of the nonlinear constraints. Those constraints will be added to the constraints already included in a rise/dsge file before being presented to the optimization function.
- **estim_linear_restrictions** [{[]}|cell]: This is most often used in the estimation of rfvar or svar models either to impose block exogeneity or to impose other forms of linear restrictions. When not empty, **estim_linear_restrictions** must be a 2-column cell: - Each row of the first column represents a particular linear combination of the estimated parameters. Those linear combinations are constructed using the **coef** class. Check help for coef.coef for more details. - Each row of the second column holds the value of the linear combination.
- **estim_blocks** [{[]}|cell]: When not empty, this triggers blockwise optimization. For further information on how to set blocks, see help for dsge.create_estimation_blocks
- **estim_priors** [{[]}|struct]: This provides an alternative to setting priors inside the rise/dsge model file. Each field of the structure must be the name of an estimated parameter. Each field will hold a cell array whose structure is described in help rise_generic.setup_priors.
- **estim_penalty** [numeric|{1e+8}]: value of the objective function when a problem occurs. Possible problems include: - no solution found - very low likelihood - stochastic singularity - problems computing the initial covariance matrix - non-positive definite covariance matrices - etc.
- **estim_cond_vars** [char|cellstr]: list of the variables to condition on during estimation of a dsge model. This then assumes that the data provided for estimation have several pages. The first page is the actual data, while the subsequent pages are the expectations data.
- **optimset** [struct]: identical to matlab's optimset
- **optimizer** [char|function handle|cell]: This can be the name of a standard matlab optimizer or RISE optimization routine or a user-defined optimization procedure available of the matlab search path. If the optimzer is provided as a cell, then the first element of the cell is the name of the optimizer or its handle and the remaining entries in the cell are additional input arguments to the user-defined optimization routine. A user-defined optimization function should have the following syntax

  ```
  [xfinal,ffinal,exitflag]=optimizer(fh,x0,lb,ub,options,varargin);
  ```

  That is, it accepts as inputs:

  - **fh**: the function to optimize
  - **x0**: a vector column of initial values of the parameters
  - **lb**: a vector column of lower bounds
  - **ub**: a vector column of upper bounds
  - **options: a structure of options whose fields will be similar** to matlab's optimset
  - **varargin: additional arguments to the user-defined** optimization procedure

  That is, it provides as outputs:

  - **xfinal**: the vector of final values
  - **ffinal**: the value of **fh** at **xfinal**
  - **exitflag**: a flag similar to the ones provided by matlab's

  optimization functions.

- **hessian_type** [{‘fd’}|’opg’]: The hessian is either computed by finite differences (fd) or by outer-product-gradient (opg)
- **hessian_repair** [{false}|true]: If the Hessian is not positive definite, it nevertheless can be repaired and prepared for a potential mcmc simulation.

5.10.3 Outputs

- **obj** [rise|dsge|rfvar|svar]: model object parameterized with the mode found and holding additional estimation results and statistics that can be found under obj.estimation

5.10.4 More About

- recursive estimation may be done easily by passing a different estim_end_date at the beginning of each estimation run.

5.10.5 Examples

See also:

Help for dsge/estimate is inherited from superclass RISE_GENERIC

5.11 filter

H1 line

5.11.1 Syntax

5.11.2 Inputs

5.11.3 Outputs

5.11.4 More About

5.11.5 Examples

See also:

5.12 forecast

forecast - computes forecasts for rise|dsge|svar|rfvar models

5.12.1 Syntax

`cond_fkst_db=forecast(obj,varargin)`

5.12.2 Inputs

- **obj** [rise|dsge|svar|rfvar]: model object
- **varargin** : additional inputs coming in pairs. These include but are not restricted to: - **forecast_to_time_series** [{true}|false]: sets the output to time

  series format or not

  - **forecast_nsteps** [integer|{12}]: number of forecasting steps
  - **forecast_start_date [char|numeric|serial date]: date when the** forecasts start (end of history + 1)
  - **forecast_conditional_hypothesis [{jma}|ncp|nas]: in dsge models in** which agents have information beyond the current period, this option determines the number of periods of shocks need to match the restrictions: - Hypothesis **jma** assumes that irrespective of how

    many periods of conditioning information are remaining, agents always receive information on the same number of shocks.

    * **Hypothesis ncp assumes there are as many shocks periods as** the number of the number of conditioning periods
    * **Hypothesis nas assumes there are as many shocks periods as** the number of anticipated steps

5.12.3 Outputs

- **cond_fkst_db** [struct|matrix]: depending on the value of **forecast_to_time_series** the returned output is a structure with time series or a cell containing a matrix and the information to reconstruct the time series.

5.12.4 More About

- the historical information as well as the conditioning information come from the same database. The time series must be organized such that for each series, the first page represents the actual data and all subsequent pages represent conditional information. If a particular condition is "nan", that location is not constrained
- Conditional forecasting for nonlinear models is also supported. However, the solving of the implied nonlinear problem may fail if the model displays instability
- Both HARD CONDITIONS and SOFT CONDITIONS are implemented but the latter are currently disabled in expectation of a better user interface.
- The data may also contain time series for a variable with name **regime** in that case, the forecast/simulation paths are computed following the information therein. **regime** must be a member of 1:h, where h is the maximum number of regimes.

5.12.5 Examples

See also: simulate

Help for dsge/forecast is inherited from superclass RISE_GENERIC

5.13 forecast_real_time

forecast_real_time - forecast from each point in time

5.13.1 Syntax

```
- [ts_fkst,ts_rmse,rmse,Updates]=forecast_real_time(obj)
- [ts_fkst,ts_rmse,rmse,Updates]=forecast_real_time(obj,varargin)
```

5.13.2 Inputs

- **obj** [dsge|svar|rfvar] : model object
- **varargin** : valid optional inputs coming in pairs. The main inputs of interest for changing the default behavior are: - **fkst_rt_nahead** [integer] : number of periods ahead

5.13.3 Outputs

- **ts_fkst** [struct] : fields are forecasts in the form of ts objects for the different endogenous variables
- **ts_rmse** [struct] : fields are RMSEs in the form of ts objects for the different endogenous variables
- **rmse** [matrix] : RMSEs for the different endogenous variables
- **Updates** [struct] : fields are the updated (in a filtering sense) in the form of ts objects for the different endogenous variables

5.13.4 More About

5.13.5 Examples

See also: plot_real_time

5.14 get

H1 line

5.14.1 Syntax

5.14.2 Inputs

5.14.3 Outputs

5.14.4 More About

5.14.5 Examples

See also:

Help for dsge/get is inherited from superclass RISE_GENERIC

5.15 historical_decomposition

historical_decomposition Computes historical decompositions of a DSGE model

5.15.1 Syntax

```
[Histdec,obj]=history_dec(obj)
[Histdec,obj]=history_dec(obj,varargin)
```

5.15.2 Inputs

- obj : [rise|dsge|rfvar|svar] model(s) for which to compute the decomposition. obj could be a vector of models
- varargin : standard optional inputs **coming in pairs**. Among which: - **histdec_start_date** : [char|numeric|{''}] : date at which the

  decomposition starts. If empty, the decomposition starts at he beginning of the history of the dataset

5.15.3 Outputs

- Histdec : [struct|cell array] structure or cell array of structures with the decompositions in each model. The decompositions are given in terms of: - the exogenous variables - **InitialConditions** : the effect of initial conditions - **risk** : measure of the effect of non-certainty equivalence - **switch** : the effect of switching (which is also a shock!!!) - **steady_state** : the contribution of the steady state

5.15.4 Remarks

- the elements that do not contribute to any of the variables are automatically discarded.
- **N.B** : a switching model is inherently nonlinear and so, strictly speaking, the type of decomposition we do for linear/linearized constant-parameter models is not feasible. RISE takes an approximation in which the variables, shocks and states matrices across states are averaged. The averaging weights are the smoothed probabilities.

5.15.5 Examples

See also:

Help for dsge/historical_decomposition is inherited from superclass RISE_GENERIC

5.16 irf

H1 line

5.16.1 Syntax

5.16.2 Inputs

5.16.3 Outputs

5.16.4 More About

5.16.5 Examples

See also:

5.17 is_stable_system

H1 line

5.17.1 Syntax

5.17.2 Inputs

5.17.3 Outputs

5.17.4 More About

5.17.5 Examples

See also:

5.18 isnan

H1 line

5.18.1 Syntax

5.18.2 Inputs

5.18.3 Outputs

5.18.4 More About

5.18.5 Examples

See also:

Help for dsge/isnan is inherited from superclass RISE_GENERIC

5.19 load_parameters

H1 line

5.19.1 Syntax

5.19.2 Inputs

5.19.3 Outputs

5.19.4 More About

5.19.5 Examples

See also:

Help for dsge/load_parameters is inherited from superclass RISE_GENERIC

5.20 log_marginal_data_density

H1 line

5.20.1 Syntax

5.20.2 Inputs

5.20.3 Outputs

5.20.4 More About

5.20.5 Examples

See also:

Help for dsge/log_marginal_data_density is inherited from superclass RISE_GENERIC

5.21 log_posterior_kernel

H1 line

5.21.1 Syntax

5.21.2 Inputs

5.21.3 Outputs

5.21.4 More About

5.21.5 Examples

See also:

Help for dsge/log_posterior_kernel is inherited from superclass RISE_GENERIC

5.22 log_prior_density

H1 line

5.22.1 Syntax

5.22.2 Inputs

5.22.3 Outputs

5.22.4 More About

5.22.5 Examples

See also:

Help for dsge/log_prior_density is inherited from superclass RISE_GENERIC

5.23 monte_carlo_filtering

H1 line

5.23.1 Syntax

5.23.2 Inputs

5.23.3 Outputs

5.23.4 More About

5.23.5 Examples

See also:

5.24 posterior_marginal_and_prior_densities

H1 line

5.24.1 Syntax

5.24.2 Inputs

5.24.3 Outputs

5.24.4 More About

5.24.5 Examples

See also:

Help for dsge/posterior_marginal_and_prior_densities is inherited from superclass RISE_GENERIC

5.25 posterior_simulator

H1 line

5.25.1 Syntax

5.25.2 Inputs

5.25.3 Outputs

5.25.4 More About

5.25.5 Examples

See also:

Help for dsge/posterior_simulator is inherited from superclass RISE_GENERIC

5.26 print_estimation_results

H1 line

5.26.1 Syntax

5.26.2 Inputs

5.26.3 Outputs

5.26.4 More About

5.26.5 Examples

See also:

Help for dsge/print_estimation_results is inherited from superclass RISE_GENERIC

5.27 print_solution

print_solution - print the solution of a model or vector of models

5.27.1 Syntax

```
print_solution(obj)
print_solution(obj,varlist)
print_solution(obj,varlist,orders)
print_solution(obj,varlist,orders,compact_form)
print_solution(obj,varlist,orders,compact_form,precision)
print_solution(obj,varlist,orders,compact_form,precision,equation_format)
print_solution(obj,varlist,orders,compact_form,precision,equation_format,file2save2)
outcell=print_solution(obj,...)
```

5.27.2 Inputs

- **obj** [rise|dsge] : model object or vector of model objects
- **varlist** [char|cellstr|{[]}] : list of variables of interest
- **orders** [numeric|{[1:solve_order]}] : orders for which we want to see the solution
- **compact_form** [{true}|false] : if true, only the solution of unique tuples (i,j,k) such that i<=j<=k is presented. If false, the solution of all combinations is presented. i.e. (i,j,k)(i,k,j)(j,i,k)(j,k,i)(k,i,j)(k,j,i)
- **precision** [char|{'%8.6f'}] : precision of the numbers printed
- **equation_format** [true|{false}] : if true, the solution is presented in the form of equations for each endogenous variable (not recommended)
- **file2save2** [char|{''}] : if not empty, the solution is written to a file rather than printed on screen. For this to happen, print_solution has to be called without ouput arguments

5.27.3 Outputs

- **outcell** [cellstr] : If an output is requested, the solution is not printed on screen or to a file.

5.27.4 More About

If a model is solved, say, up to 3rd order, one may still want to see the first-order solution or the solution up to second-order only or any combination of orders.

5.27.5 Examples

See also:

5.28 prior_plots

H1 line

5.28.1 Syntax

5.28.2 Inputs

5.28.3 Outputs

5.28.4 More About

5.28.5 Examples

See also:

Help for dsge/prior_plots is inherited from superclass RISE_GENERIC

5.29 report

REPORT assigns the elements of interest to a rise_report.report object

5.29.1 Syntax

::

- REPORT(rise.empty(0)) : displays the default inputs
- REPORT(obj,destination_root,rep_items) : assigns the reported elements in rep_items to destination_root
- REPORT(obj,destination_root,rep_items,varargin) : assigns varargin to obj before doing the rest

5.29.2 Inputs

- obj : [rise|dsge]
- destination_root : [rise_report.report] : handle for the actual report
- rep_items : [char|cellstr] : list of desired items to report. This list can only include : 'endogenous', 'exogenous', 'observables', 'parameters', 'solution', 'estimation', 'estimation_statistics', 'equations', 'code'

5.29.3 Outputs

none

5.29.4 More About

5.29.5 Examples

See also:

Help for dsge/report is inherited from superclass RISE_GENERIC

5.30 resid

H1 line

5.30.1 Syntax

5.30.2 Inputs

5.30.3 Outputs

5.30.4 More About

5.30.5 Examples

See also:

5.31 set

set - sets options for dsge|rise models

5.31.1 Syntax

```
obj=set(obj,varargin)
```

5.31.2 Inputs

- **obj** [rise|dsge]: model object
- **varargin** : valid input arguments coming in pairs. Notable fields to that can be set include and are not restricted to: - **solve_shock_horizon** [integer|struct|cell]
  - for the integer case, all shocks are set to the same integer
  - **for the struct case, the input must be a structure with shock** names as fields. Only the shock names whose value is to change have to be listed. In this case, different shocks can have different horizons k. The default is k=0 i.e. agents don't see into the future
  - **for the cell case, the cell should have two columns. The first** column includes the names of the shocks whose horizon is to change. The second column includes the horizon for each shock name on the left.
  - **solve_function_mode [{explicit/amateur}|vectorized/professional|disc]**
    - **in the amateur or explicit mode the functions are kept in** cell arrays of anonymous functions and evaluated using for loops
    - **in the vectorized or professional mode the functions are** compacted into one long and unreadable function.
    - **in the disc mode the functions are written to disc in a** subdirectory called routines.

5.31.3 Outputs

- **obj** [rise|dsge]: model object

5.31.4 More About

5.31.5 Examples

obj=set(obj,'solve_shock_horizon',struct('shock1',2,'shock3',4)) obj=set(obj,'solve_shock_horizon',5)

See also: rise_generic.set

5.32 set_solution_to_companion

H1 line

5.32.1 Syntax

5.32.2 Inputs

5.32.3 Outputs

5.32.4 More About

5.32.5 Examples

See also:

5.33 simulate

simulate - simulates a RISE model

5.33.1 Syntax

```
[db,states,retcode] = simulate(obj,varargin)
```

5.33.2 Inputs

- **obj** [rfvar|dsge|rise|svar]: model object
- **varargin** : additional arguments including but not restricted to
  - **simul_periods** [integer|{100}]: number of simulation periods
  - **simul_burn** [integer|{100}]: number of burn-in periods

- **simul_algo [[{mt19937ar}| mcg16807|mlfg6331_64|mrg32k3a|** shr3cong|swb2712]]: matlab's seeding algorithms
- **simul_seed** [numeric|{0}]: seed of the computations
- **simul_historical_data [ts|struct|{''}]: historical data from** which the simulations are based. If empty, the simulations start at the steady state.
- **simul_history_end_date [char|integer|serial date]: last date of** history
- **simul_regime [integer|vector|{[]}]: regimes for which the model** is simulated
- **simul_update_shocks_handle [function handle]: we may want to** update the shocks if some condition on the state of the economy is satisfied. For instance, shock monetary policy to keep the interest rate at the floor for an extented period of time if we already are at the ZLB/ZIF. simul_update_shocks_handle takes as inputs the current shocks and the state vector (all the endogenous variables) and returns the updated shocks. But for all this to be put into motion, the user also has to turn on **simul_do_update_shocks** by setting it to true.
- **simul_do_update_shocks [true|{false}]: update the shocks based on simul_update_shocks_handle** or not.
- **simul_to_time_series [{true}|false]: if true, the output is a** time series, else a cell array with a matrix and information on elements that help reconstruct the time series.

5.33.3 Outputs

- **db** [struct|cell array]: if **simul_to_time_series** is true, the output is a time series, else a cell array with a matrix and information on elements that help reconstruct the time series.
- **states** [vector]: history of the regimes over the forecast horizon
- **retcode** [integer]: if 0, the simulation went fine. Else something got wrong. In that case one can understand the problem by running decipher(retcode)

5.33.4 More About

- **simul_historical_data** contains the historical data as well as conditional information over the forecast horizon. It may also include as an alternative to **simul_regime**, a time series with name **regime**, which indicates the regimes over the forecast horizon.

5.33.5 Examples

See also:

Help for dsge/simulate is inherited from superclass RISE_GENERIC

5.34 simulate_nonlinear

H1 line

5.34.1 Syntax

5.34.2 Inputs

5.34.3 Outputs

5.34.4 More About

5.34.5 Examples

See also:

5.35 simulation_diagnostics

H1 line

5.35.1 Syntax

5.35.2 Inputs

5.35.3 Outputs

5.35.4 More About

5.35.5 Examples

See also:

Help for dsge/simulation_diagnostics is inherited from superclass RISE_GENERIC

5.36 solve

H1 line

5.36.1 Syntax

5.36.2 Inputs

5.36.3 Outputs

5.36.4 More About

5.36.5 Examples

obj=solve(obj,'solve_shock_horizon',struct('shock1',2,'shock3',4)) obj=solve(obj,'solve_shock_horizon',5)

See also:

5.37 solve_alternatives

H1 line

5.37.1 Syntax

5.37.2 Inputs

5.37.3 Outputs

5.37.4 More About

5.37.5 Examples

See also:

5.38 stoch_simul

H1 line

5.38.1 Syntax

5.38.2 Inputs

5.38.3 Outputs

5.38.4 More About

5.38.5 Examples

See also:

Help for dsge/stoch_simul is inherited from superclass RISE_GENERIC

5.39 theoretical_autocorrelations

H1 line

5.39.1 Syntax

5.39.2 Inputs

5.39.3 Outputs

5.39.4 More About

5.39.5 Examples

See also:

Help for dsge/theoretical_autocorrelations is inherited from superclass RISE_GENERIC

5.40 theoretical_autocovariances

H1 line

5.40.1 Syntax

5.40.2 Inputs

5.40.3 Outputs

5.40.4 More About

5.40.5 Examples

See also:

Help for dsge/theoretical_autocovariances is inherited from superclass RISE_GENERIC

5.41 variance_decomposition

H1 line

5.41.1 Syntax

5.41.2 Inputs

5.41.3 Outputs

5.41.4 More About

5.41.5 Examples

See also:

Help for dsge/variance_decomposition is inherited from superclass RISE_GENERIC

CHAPTER

SIX

REDUCED-FORM VAR MODELING

6.1 methods

- [check_identification](rfvar/check_identification)
- [check_optimum](rfvar/check_optimum)
- [draw_parameter](rfvar/draw_parameter)
- [estimate](rfvar/estimate)
- [forecast](rfvar/forecast)
- [get](rfvar/get)
- [historical_decomposition](rfvar/historical_decomposition)
- [irf](rfvar/irf)
- [isnan](rfvar/isnan)
- [load_parameters](rfvar/load_parameters)
- [log_marginal_data_density](rfvar/log_marginal_data_density)
- [log_posterior_kernel](rfvar/log_posterior_kernel)
- [log_prior_density](rfvar/log_prior_density)
- [msvar_priors](rfvar/msvar_priors)
- [posterior_marginal_and_prior_densities](rfvar/posterior_marginal_and_prior_densities)
- [posterior_simulator](rfvar/posterior_simulator)
- [print_estimation_results](rfvar/print_estimation_results)
- [prior_plots](rfvar/prior_plots)
- [report](rfvar/report)
- [rfvar](rfvar/rfvar)
- [set](rfvar/set)
- [set_solution_to_companion](rfvar/set_solution_to_companion)
- [simulate](rfvar/simulate)
- [simulation_diagnostics](rfvar/simulation_diagnostics)
- [solve](rfvar/solve)

- [stoch_simul](rfvar/stoch_simul)
- [structural_form](rfvar/structural_form)
- [template](rfvar/template)
- [theoretical_autocorrelations](rfvar/theoretical_autocorrelations)
- [theoretical_autocovariances](rfvar/theoretical_autocovariances)
- [variance_decomposition](rfvar/variance_decomposition)

6.2 properties

- [identification] -
- [structural_shocks] -
- [nonlinear_restrictions] -
- [constant] -
- [nlags] -
- [legend] -
- [endogenous] -
- [exogenous] -
- [parameters] -
- [observables] -
- [markov_chains] -
- [options] -
- [estimation] -
- [solution] -
- [filtering] -

6.3 check_identification

H1 line

6.3.1 Syntax

6.3.2 Inputs

6.3.3 Outputs

6.3.4 More About

6.3.5 Examples

See also:

6.4 check_optimum

H1 line

6.4.1 Syntax

6.4.2 Inputs

6.4.3 Outputs

6.4.4 More About

6.4.5 Examples

See also:

Help for rfvar/check_optimum is inherited from superclass RISE_GENERIC

6.5 draw_parameter

H1 line

6.5.1 Syntax

6.5.2 Inputs

6.5.3 Outputs

6.5.4 More About

6.5.5 Examples

See also:

Help for rfvar/draw_parameter is inherited from superclass RISE_GENERIC

6.6 estimate

estimate - estimates the parameters of a RISE model

6.6.1 Syntax

```
obj=estimate(obj)
obj=estimate(obj,varargin)
```

6.6.2 Inputs

- **obj** [rise|dsge|rfvar|svar]: model object
- **varargin** additional optional inputs among which the most relevant for estimation are:
- **estim_parallel** [integer|{1}]: Number of starting values
- **estim_start_from_mode** [true|false|{[]}]: when empty, the user is prompted to answer the question as to whether to start estimation from a previously found mode or not. If true or false, no question is asked.
- **estim_start_date** [numeric|char|serial date]: date of the first observation to use in the dataset provided for estimation
- **estim_end_date** [numeric|char|serial date]: date of the last observation to use in the dataset provided for estimation
- **estim_max_trials** [integer|{500}]: When the initial value of the log-likelihood is too low, RISE uniformly draws from the prior support in search for a better starting point. It will try this for a maximum number of **estim_max_trials** times before squeaking with an error.
- **estim_start_vals** [{[]}|struct]: when not empty, the parameters whose names are fields of the structure will see their start values updated or overriden by the information in **estim_start_vals**. There is no need to provide values to update the start values for the estimated parameters.
- **estim_general_restrictions** [{[]}|function handle]: when not empty, the argument should be a function handle that takes as input a parameterized RISE object and returns a scalar or vector of numbers representing the strength of the violation of the nonlinear constraints. Those constraints will be added to the constraints already included in a rise/dsge file before being presented to the optimization function.
- **estim_linear_restrictions** [{[]}|cell]: This is most often used in the estimation of rfvar or svar models either to impose block exogeneity or to impose other forms of linear restrictions. When not empty, **estim_linear_restrictions** must be a 2-column cell: - Each row of the first column represents a particular linear combination of the estimated parameters. Those linear combinations are constructed using the **coef** class. Check help for coef.coef for more details. - Each row of the second column holds the value of the linear combination.
- **estim_blocks** [{[]}|cell]: When not empty, this triggers blockwise optimization. For further information on how to set blocks, see help for dsge.create_estimation_blocks
- **estim_priors** [{[]}|struct]: This provides an alternative to setting priors inside the rise/dsge model file. Each field of the structure must be the name of an estimated parameter. Each field will hold a cell array whose structure is described in help rise_generic.setup_priors.

- **estim_penalty** [numeric|{1e+8}]: value of the objective function when a problem occurs. Possible problems include: - no solution found - very low likelihood - stochastic singularity - problems computing the initial covariance matrix - non-positive definite covariance matrices - etc.
- **estim_cond_vars** [char|cellstr]: list of the variables to condition on during estimation of a dsge model. This then assumes that the data provided for estimation have several pages. The first page is the actual data, while the subsequent pages are the expectations data.
- **optimset** [struct]: identical to matlab's optimset
- **optimizer** [char|function handle|cell]: This can be the name of a standard matlab optimizer or RISE optimization routine or a user-defined optimization procedure available of the matlab search path. If the optimzer is provided as a cell, then the first element of the cell is the name of the optimizer or its handle and the remaining entries in the cell are additional input arguments to the user-defined optimization routine. A user-defined optimization function should have the following syntax

  ```
  [xfinal,ffinal,exitflag]=optimizer(fh,x0,lb,ub,options,varargin);
  ```

  That is, it accepts as inputs:

  - **fh**: the function to optimize
  - **x0**: a vector column of initial values of the parameters
  - **lb**: a vector column of lower bounds
  - **ub**: a vector column of upper bounds
  - **options: a structure of options whose fields will be similar** to matlab's optimset
  - **varargin: additional arguments to the user-defined** optimization procedure

  That is, it provides as outputs:

  - **xfinal**: the vector of final values
  - **ffinal**: the value of **fh** at **xfinal**
  - **exitflag**: a flag similar to the ones provided by matlab's

  optimization functions.

- **hessian_type** [{'fd'}|'opg']: The hessian is either computed by finite differences (fd) or by outer-product-gradient (opg)
- **hessian_repair** [{false}|true]: If the Hessian is not positive definite, it nevertheless can be repaired and prepared for a potential mcmc simulation.

6.6.3 Outputs

- **obj** [rise|dsge|rfvar|svar]: model object parameterized with the mode found and holding additional estimation results and statistics that can be found under obj.estimation

6.6.4 More About

- recursive estimation may be done easily by passing a different estim_end_date at the beginning of each estimation run.

6.6.5 Examples

See also:

Help for rfvar/estimate is inherited from superclass RISE_GENERIC

6.7 forecast

forecast - computes forecasts for rise|dsge|svar|rfvar models

6.7.1 Syntax

```
cond_fkst_db=forecast(obj,varargin)
```

6.7.2 Inputs

- **obj** [rise|dsge|svar|rfvar]: model object
- **varargin** : additional inputs coming in pairs. These include but are not restricted to: - **forecast_to_time_series** [{true}|false]: sets the output to time

  series format or not

  - **forecast_nsteps** [integer|{12}]: number of forecasting steps
  - **forecast_start_date [char|numeric|serial date]: date when the** forecasts start (end of history + 1)
  - **forecast_conditional_hypothesis [{jma}|ncp|nas]: in dsge models in** which agents have information beyond the current period, this option determines the number of periods of shocks need to match the restrictions: - Hypothesis **jma** assumes that irrespective of how

    many periods of conditioning information are remaining, agents always receive information on the same number of shocks.

    - **Hypothesis ncp assumes there are as many shocks periods as** the number of the number of conditioning periods
    - **Hypothesis nas assumes there are as many shocks periods as** the number of anticipated steps

6.7.3 Outputs

- **cond_fkst_db** [struct|matrix]: depending on the value of **forecast_to_time_series** the returned output is a structure with time series or a cell containing a matrix and the information to reconstruct the time series.

6.7.4 More About

- the historical information as well as the conditioning information come from the same database. The time series must be organized such that for each series, the first page represents the actual data and all subsequent pages represent conditional information. If a particular condition is "nan", that location is not constrained
- Conditional forecasting for nonlinear models is also supported. However, the solving of the implied nonlinear problem may fail if the model displays instability

- Both HARD CONDITIONS and SOFT CONDITIONS are implemented but the latter are currently disabled in expectation of a better user interface.
- The data may also contain time series for a variable with name **regime** in that case, the forecast/simulation paths are computed following the information therein. **regime** must be a member of 1:h, where h is the maximum number of regimes.

6.7.5 Examples

See also: simulate

Help for rfvar/forecast is inherited from superclass RISE_GENERIC

6.8 get

H1 line

6.8.1 Syntax

6.8.2 Inputs

6.8.3 Outputs

6.8.4 More About

6.8.5 Examples

See also:

Help for rfvar/get is inherited from superclass RISE_GENERIC

6.9 historical_decomposition

historical_decomposition Computes historical decompositions of a DSGE model

6.9.1 Syntax

```
[Histdec,obj]=history_dec(obj)
[Histdec,obj]=history_dec(obj,varargin)
```

6.9.2 Inputs

- obj : [rise|dsge|rfvar|svar] model(s) for which to compute the decomposition. obj could be a vector of models
- varargin : standard optional inputs **coming in pairs**. Among which: - **histdec_start_date** : [char|numeric|{''}] : date at which the

    decomposition starts. If empty, the decomposition starts at he beginning of the history of the dataset

6.9.3 Outputs

- Histdec : [struct|cell array] structure or cell array of structures with the decompositions in each model. The decompositions are given in terms of: - the exogenous variables - **InitialConditions** : the effect of initial conditions - **risk** : measure of the effect of non-certainty equivalence - **switch** : the effect of switching (which is also a shock!!!) - **steady_state** : the contribution of the steady state

6.9.4 Remarks

- the elements that do not contribute to any of the variables are automatically discarded.
- **N.B** : a switching model is inherently nonlinear and so, strictly speaking, the type of decomposition we do for linear/linearized constant-parameter models is not feasible. RISE takes an approximation in which the variables, shocks and states matrices across states are averaged. The averaging weights are the smoothed probabilities.

6.9.5 Examples

See also:

Help for rfvar/historical_decomposition is inherited from superclass RISE_GENERIC

6.10 irf

irf - computes impulse responses for a RISE model

6.10.1 Syntax

```
myirfs=irf(obj)

myirfs=irf(obj,varargin)
```

6.10.2 Inputs

- **obj** [rise|dsge|rfvar|svar]: single or vector of RISE models
- **varargin** : optional options coming in pairs. The notable ones that will influence the behavior of the impulse responses are:
- **irf_shock_list** [char|cellstr|{''}]: list of shocks for which we want to compute impulse responses
- **irf_var_list** [char|cellstr|{''}]: list of the endogenous variables we want to report

- **irf_periods** [integer|{40}]: length of the irfs
- **irf_shock_sign** [numeric|-1|{1}]: sign or scale of the original impulse. If **irf_shock_sign** >0, we get impulse responses to a positive shock. If **irf_shock_sign** <0, the responses are negative. If If **irf_shock_sign** =0, all the responses are 0.
- **irf_draws** [integer|{50}]: number of draws used in the simulation impulse responses in a nonlinear model. A nonlinear model is defined as a model that satisfies at least one of the following criteria - solved at an order >1 - has more than one regime and option **irf_regime_specific** below is

  set to false
- **irf_type** [{irf}|girf]: type of irfs. If the type is irf, the impulse responses are computed directly exploiting the fact that the model is linear. If the type is girf, the formula for the generalized impulse responses is used: the irf is defined as the expectation of the difference of two simulation paths. In the first path the initial impulse for the shock of interest is nonzero while it is zero for the second path. All other shocks are the same for both paths in a given simulation.
- **irf_regime_specific** [{true}|false]: In a switching model, we may or may not want to compute impulse responses specific to each regime.
- **irf_use_historical_data** [{false}|true]: if true, the data stored in option **simul_historical_data** are used as initial conditions. But the model has to be nonlinear otherwise the initial conditions are set to zero. This option gives the flexibility to set the initial conditions for the impulse responses.
- **irf_to_time_series** [{true}|false]: If true, the output is in the form of time series. Else it is in the form of a cell containing the information needed to reconstruct the time series.

6.10.3 Outputs

- **myirfs** [{struct}|cell]: Impulse response data

6.10.4 More About

- for linear models or models solved up to first order, the initial conditions as well as the steady states are set to 0 in the computation of the impulse responses.
- for nonlinear models, the initial conditions is the ergodic mean

6.10.5 Examples

See also:

Help for rfvar/irf is inherited from superclass RISE_GENERIC

6.11 isnan

H1 line

6.11.1 Syntax

6.11.2 Inputs

6.11.3 Outputs

6.11.4 More About

6.11.5 Examples

See also:

Help for rfvar/isnan is inherited from superclass RISE_GENERIC

6.12 load_parameters

H1 line

6.12.1 Syntax

6.12.2 Inputs

6.12.3 Outputs

6.12.4 More About

6.12.5 Examples

See also:

Help for rfvar/load_parameters is inherited from superclass RISE_GENERIC

6.13 log_marginal_data_density

H1 line

6.13.1 Syntax

6.13.2 Inputs

6.13.3 Outputs

6.13.4 More About

6.13.5 Examples

See also:

Help for rfvar/log_marginal_data_density is inherited from superclass RISE_GENERIC

6.14 log_posterior_kernel

H1 line

6.14.1 Syntax

6.14.2 Inputs

6.14.3 Outputs

6.14.4 More About

6.14.5 Examples

See also:

Help for rfvar/log_posterior_kernel is inherited from superclass RISE_GENERIC

6.15 log_prior_density

H1 line

6.15.1 Syntax

6.15.2 Inputs

6.15.3 Outputs

6.15.4 More About

6.15.5 Examples

See also:

Help for rfvar/log_prior_density is inherited from superclass RISE_GENERIC

6.16 msvar_priors

H1 line

6.16.1 Syntax

6.16.2 Inputs

6.16.3 Outputs

6.16.4 More About

6.16.5 Examples

See also:

Help for rfvar/msvar_priors is inherited from superclass SVAR

6.17 posterior_marginal_and_prior_densities

H1 line

6.17.1 Syntax

6.17.2 Inputs

6.17.3 Outputs

6.17.4 More About

6.17.5 Examples

See also:

Help for rfvar/posterior_marginal_and_prior_densities is inherited from superclass RISE_GENERIC

6.18 posterior_simulator

H1 line

6.18.1 Syntax

6.18.2 Inputs

6.18.3 Outputs

6.18.4 More About

6.18.5 Examples

See also:

Help for rfvar/posterior_simulator is inherited from superclass RISE_GENERIC

6.19 print_estimation_results

H1 line

6.19.1 Syntax

6.19.2 Inputs

6.19.3 Outputs

6.19.4 More About

6.19.5 Examples

See also:

Help for rfvar/print_estimation_results is inherited from superclass RISE_GENERIC

6.20 prior_plots

H1 line

6.20.1 Syntax

6.20.2 Inputs

6.20.3 Outputs

6.20.4 More About

6.20.5 Examples

See also:

Help for rfvar/prior_plots is inherited from superclass RISE_GENERIC

6.21 report

REPORT assigns the elements of interest to a rise_report.report object

6.21.1 Syntax

::

- REPORT(rise.empty(0)) : displays the default inputs
- REPORT(obj,destination_root,rep_items) : assigns the reported elements in rep_items to destination_root
- REPORT(obj,destination_root,rep_items,varargin) : assigns varargin to obj before doing the rest

6.21.2 Inputs

- obj : [rise|dsge]
- destination_root : [rise_report.report] : handle for the actual report
- rep_items : [char|cellstr] : list of desired items to report. This list can only include : 'endogenous', 'exogenous', 'observables', 'parameters', 'solution', 'estimation', 'estimation_statistics', 'equations', 'code'

6.21.3 Outputs

none

6.21.4 More About

6.21.5 Examples

See also:

Help for rfvar/report is inherited from superclass RISE_GENERIC

6.22 rfvar

~~ no help found

6.23 set

set - sets options for RISE models

6.23.1 Syntax

```
obj=set(obj,varargin)
```

6.23.2 Inputs

- **obj** [rise|dsge|svar|rfvar]: model object
- **varargin** : valid input arguments coming in pairs.

6.23.3 Outputs

- **obj** [rise|dsge|svar|rfvar]: model object

6.23.4 More About

- one can force a new field into the options by prefixing it with a '+' sign. Let's say yourfield is not part of the options and you would like to force it to be in the options because it is going to be used in some function or algorithm down the road. Then you can run m=set(m,'+yourfield',value). then m will be part of the new options.

6.23.5 Examples

See also:

Help for rfvar/set is inherited from superclass RISE_GENERIC

6.24 set_solution_to_companion

H1 line

6.24.1 Syntax

6.24.2 Inputs

6.24.3 Outputs

6.24.4 More About

6.24.5 Examples

See also:

Help for rfvar/set_solution_to_companion is inherited from superclass SVAR

6.25 simulate

simulate - simulates a RISE model

6.25.1 Syntax

```
[db,states,retcode] = simulate(obj,varargin)
```

6.25.2 Inputs

- **obj** [rfvar|dsge|rise|svar]: model object
- **varargin** : additional arguments including but not restricted to
  - **simul_periods** [integer|{100}]: number of simulation periods
  - **simul_burn** [integer|{100}]: number of burn-in periods
  - **simul_algo [[{mt19937ar}| mcg16807|mlfg6331_64|mrg32k3a|** shr3cong|swb2712]]: matlab's seeding algorithms
  - **simul_seed** [numeric|{0}]: seed of the computations
  - **simul_historical_data [ts|struct|{''}]: historical data from** which the simulations are based. If empty, the simulations start at the steady state.
  - **simul_history_end_date [char|integer|serial date]: last date of** history
  - **simul_regime [integer|vector|{[]}]: regimes for which the model** is simulated
  - **simul_update_shocks_handle [function handle]: we may want to** update the shocks if some condition on the state of the economy is satisfied. For instance, shock monetary policy to keep the interest rate at the floor for an extented period of time if we already are at the ZLB/ZIF. simul_update_shocks_handle takes as inputs the current shocks and the state vector (all the endogenous variables) and returns the updated shocks. But for all this to be put into motion, the user also has to turn on **simul_do_update_shocks** by setting it to true.
  - **simul_do_update_shocks [true|{false}]: update the shocks based on simul_update_shocks_handle** or not.
  - **simul_to_time_series [{true}|false]: if true, the output is a** time series, else a cell array with a matrix and information on elements that help reconstruct the time series.

6.25.3 Outputs

- **db** [struct|cell array]: if **simul_to_time_series** is true, the output is a time series, else a cell array with a matrix and information on elements that help reconstruct the time series.
- **states** [vector]: history of the regimes over the forecast horizon
- **retcode** [integer]: if 0, the simulation went fine. Else something got wrong. In that case one can understand the problem by running decipher(retcode)

6.25.4 More About

- **simul_historical_data** contains the historical data as well as conditional information over the forecast horizon. It may also include as an alternative to **simul_regime**, a time series with name **regime**, which indicates the regimes over the forecast horizon.

6.25.5 Examples

See also:

Help for rfvar/simulate is inherited from superclass RISE_GENERIC

6.26 simulation_diagnostics

H1 line

6.26.1 Syntax

6.26.2 Inputs

6.26.3 Outputs

6.26.4 More About

6.26.5 Examples

See also:

Help for rfvar/simulation_diagnostics is inherited from superclass RISE_GENERIC

6.27 solve

H1 line

6.27.1 Syntax

6.27.2 Inputs

6.27.3 Outputs

6.27.4 More About

6.27.5 Examples

See also:

6.28 stoch_simul

H1 line

6.28.1 Syntax

6.28.2 Inputs

6.28.3 Outputs

6.28.4 More About

6.28.5 Examples

See also:

Help for rfvar/stoch_simul is inherited from superclass RISE_GENERIC

6.29 structural_form

structural_form finds A structural form given the imposed restrictions

6.29.1 Syntax

```
newobj=structural_form(obj)
newobj=structural_form(obj,varargin)
```

6.29.2 Inputs

- **obj** : [rfvar] : reduced form VAR object
- varargin : standard optional inputs **coming in pairs**. Among which:
  - **restrict_lags : [cell array|{‘’}] : restrictions on the lag** structure. There are two equivalent syntaxes for this:
    - {‘var_name1@var_name2{lag}’}
    - {‘alag(var_name1,var_name2)’} : here alag should be understood as a-lag, where lag is the “lag” e.g. a1(infl,unemp) means unemp does not enter the infl equation at lag 1.
  - **restrict_irf_sign : [cell array|{‘’}] : sign restrictions on the** impulse responses. The general syntax is {‘var_name{period}@shock_name‘,’sign’} and the default period is “0” (for contemporaneous). That means {‘var_name{0}@shock_name‘,’+’} and {‘var_name@shock_name‘,’+’} are equivalent
  - **restrict_irf_zero : [cell array|{‘’}] : zero restrictions on the** impulse responses. The general syntax is {‘var_name{period}@shock_name‘} and the default period is “0” (for contemporaneous). That means {‘var_name{0}@shock_name‘} and {‘var_name@shock_name‘} are equivalent
  - **structural_shocks : [cell array|{‘’}] : List of structural** shocks. The shock names can be entered with or without their description. For instance : - {‘E_PAI’,’E_U’,’E_MP’} - {‘E_PAI’,“‘inflation shock”’,’E_U’,“‘unempl shock”’,’E_MP’}
  - **irf_sample_max** [[numeric|{10000}]][maximum number of trials in] the drawing of rotation matrices

6.29.3 Outputs

- newobj : [rfvar]: new rfvar object with the drawn structural form

6.29.4 More About

- RISE automatically orders the endogenous variables alphabetically and tags each equation with one of the endogenous variables. This may be useful for understanding the behavior of **restrict_lags** above.
- The Choleski identification scheme is not implemented per se. The user has to explicitly enter the zeros in the right places. This gives the flexibility in implementing the restrictions. For instance, one could imagine a scheme in which choleski restrictions hold only in the long run.
- With only zero restrictions, one cannot expect the impulse responses to automatically have the correct sign. The rotation imposes zero restrictions but not the sign. If you would like to have correctly-signed impulse responses there are two choices: - explicitly add sign restrictions - multiply the impulse responses for the wrongly-signed shock with minus.
- If the signs are not explicitly enforced under zeros restrictions, in an exercise in which one draws many rotations, some will have one sign and some others a diffferent sign. Here, perhaps more than elsewhere, it is important to add some sign restrictions to have consistent results throughout.
- Many periods can be entered simultaneously. For instance 'var_name{0,3,5,10:20,inf}@shock_name'
- long-run restrictions are denoted by "inf". For instance 'var_name{inf}@shock_name'
- Identification for Markov switching VARs is not implemented/supported.

6.29.5 Examples

See also:

6.30 template

~~ no help found

6.31 theoretical_autocorrelations

H1 line

6.31.1 Syntax

6.31.2 Inputs

6.31.3 Outputs

6.31.4 More About

6.31.5 Examples

See also:

Help for rfvar/theoretical_autocorrelations is inherited from superclass RISE_GENERIC

6.32 theoretical_autocovariances

H1 line

6.32.1 Syntax

6.32.2 Inputs

6.32.3 Outputs

6.32.4 More About

6.32.5 Examples

See also:

Help for rfvar/theoretical_autocovariances is inherited from superclass RISE_GENERIC

6.33 variance_decomposition

H1 line

6.33.1 Syntax

6.33.2 Inputs

6.33.3 Outputs

6.33.4 More About

6.33.5 Examples

See also:

Help for rfvar/variance_decomposition is inherited from superclass RISE_GENERIC

CHAPTER

SEVEN

STRUCTURAL VAR MODELING

7.1 methods

- [check_optimum](svar/check_optimum)
- [draw_parameter](svar/draw_parameter)
- [estimate](svar/estimate)
- [forecast](svar/forecast)
- [get](svar/get)
- [historical_decomposition](svar/historical_decomposition)
- [irf](svar/irf)
- [isnan](svar/isnan)
- [load_parameters](svar/load_parameters)
- [log_marginal_data_density](svar/log_marginal_data_density)
- [log_posterior_kernel](svar/log_posterior_kernel)
- [log_prior_density](svar/log_prior_density)
- [msvar_priors](svar/msvar_priors)
- [posterior_marginal_and_prior_densities](svar/posterior_marginal_and_prior_densities)
- [posterior_simulator](svar/posterior_simulator)
- [print_estimation_results](svar/print_estimation_results)
- [prior_plots](svar/prior_plots)
- [report](svar/report)
- [set](svar/set)
- [set_solution_to_companion](svar/set_solution_to_companion)
- [simulate](svar/simulate)
- [simulation_diagnostics](svar/simulation_diagnostics)
- [solve](svar/solve)
- [stoch_simul](svar/stoch_simul)
- [svar](svar/svar)

- [template](svar/template)
- [theoretical_autocorrelations](svar/theoretical_autocorrelations)
- [theoretical_autocovariances](svar/theoretical_autocovariances)
- [variance_decomposition](svar/variance_decomposition)

7.2 properties

- [constant] -
- [nlags] -
- [legend] -
- [endogenous] -
- [exogenous] -
- [parameters] -
- [observables] -
- [markov_chains] -
- [options] -
- [estimation] -
- [solution] -
- [filtering] -

7.3 check_optimum

H1 line

7.3.1 Syntax

7.3.2 Inputs

7.3.3 Outputs

7.3.4 More About

7.3.5 Examples

See also:

Help for svar/check_optimum is inherited from superclass RISE_GENERIC

7.4 draw_parameter

H1 line

7.4.1 Syntax

7.4.2 Inputs

7.4.3 Outputs

7.4.4 More About

7.4.5 Examples

See also:

Help for svar/draw_parameter is inherited from superclass RISE_GENERIC

7.5 estimate

estimate - estimates the parameters of a RISE model

7.5.1 Syntax

```
obj=estimate(obj)
obj=estimate(obj,varargin)
```

7.5.2 Inputs

- **obj** [rise|dsge|rfvar|svar]: model object
- **varargin** additional optional inputs among which the most relevant for estimation are:
- **estim_parallel** [integer|{1}]: Number of starting values
- **estim_start_from_mode** [true|false|{[]}]: when empty, the user is prompted to answer the question as to whether to start estimation from a previously found mode or not. If true or false, no question is asked.
- **estim_start_date** [numeric|char|serial date]: date of the first observation to use in the dataset provided for estimation
- **estim_end_date** [numeric|char|serial date]: date of the last observation to use in the dataset provided for estimation
- **estim_max_trials** [integer|{500}]: When the initial value of the log-likelihood is too low, RISE uniformly draws from the prior support in search for a better starting point. It will try this for a maximum number of **estim_max_trials** times before squeaking with an error.
- **estim_start_vals** [{[]}|struct]: when not empty, the parameters whose names are fields of the structure will see their start values updated or overriden by the information in **estim_start_vals**. There is no need to provide values to update the start values for the estimated parameters.

- **estim_general_restrictions** [{[]}|function handle]: when not empty, the argument should be a function handle that takes as input a parameterized RISE object and returns a scalar or vector of numbers representing the strength of the violation of the nonlinear constraints. Those constraints will be added to the constraints already included in a rise/dsge file before being presented to the optimization function.
- **estim_linear_restrictions** [{[]}|cell]: This is most often used in the estimation of rfvar or svar models either to impose block exogeneity or to impose other forms of linear restrictions. When not empty, **estim_linear_restrictions** must be a 2-column cell: - Each row of the first column represents a particular linear combination of the estimated parameters. Those linear combinations are constructed using the **coef** class. Check help for coef.coef for more details. - Each row of the second column holds the value of the linear combination.
- **estim_blocks** [{[]}|cell]: When not empty, this triggers blockwise optimization. For further information on how to set blocks, see help for dsge.create_estimation_blocks
- **estim_priors** [{[]}|struct]: This provides an alternative to setting priors inside the rise/dsge model file. Each field of the structure must be the name of an estimated parameter. Each field will hold a cell array whose structure is described in help rise_generic.setup_priors.
- **estim_penalty** [numeric|{1e+8}]: value of the objective function when a problem occurs. Possible problems include: - no solution found - very low likelihood - stochastic singularity - problems computing the initial covariance matrix - non-positive definite covariance matrices - etc.
- **estim_cond_vars** [char|cellstr]: list of the variables to condition on during estimation of a dsge model. This then assumes that the data provided for estimation have several pages. The first page is the actual data, while the subsequent pages are the expectations data.
- **optimset** [struct]: identical to matlab's optimset
- **optimizer** [char|function handle|cell]: This can be the name of a standard matlab optimizer or RISE optimization routine or a user-defined optimization procedure available of the matlab search path. If the optimzer is provided as a cell, then the first element of the cell is the name of the optimizer or its handle and the remaining entries in the cell are additional input arguments to the user-defined optimization routine. A user-defined optimization function should have the following syntax

  ```
  [xfinal,ffinal,exitflag]=optimizer(fh,x0,lb,ub,options,varargin);
  ```

  That is, it accepts as inputs:

  - **fh**: the function to optimize
  - **x0**: a vector column of initial values of the parameters
  - **lb**: a vector column of lower bounds
  - **ub**: a vector column of upper bounds
  - **options: a structure of options whose fields will be similar** to matlab's optimset
  - **varargin: additional arguments to the user-defined** optimization procedure

  That is, it provides as outputs:

  - **xfinal**: the vector of final values
  - **ffinal**: the value of **fh** at **xfinal**
  - **exitflag**: a flag similar to the ones provided by matlab's

  optimization functions.
- **hessian_type** [{'fd'}|'opg']: The hessian is either computed by finite differences (fd) or by outer-product-gradient (opg)
- **hessian_repair** [{false}|true]: If the Hessian is not positive definite, it nevertheless can be repaired and prepared for a potential mcmc simulation.

7.5.3 Outputs

- **obj** [rise|dsge|rfvar|svar]: model object parameterized with the mode found and holding additional estimation results and statistics that can be found under obj.estimation

7.5.4 More About

- recursive estimation may be done easily by passing a different estim_end_date at the beginning of each estimation run.

7.5.5 Examples

See also:

Help for svar/estimate is inherited from superclass RISE_GENERIC

7.6 forecast

forecast - computes forecasts for rise|dsge|svar|rfvar models

7.6.1 Syntax

```
cond_fkst_db=forecast(obj,varargin)
```

7.6.2 Inputs

- **obj** [rise|dsge|svar|rfvar]: model object
- **varargin** : additional inputs coming in pairs. These include but are not restricted to: - **forecast_to_time_series** [{true}|false]: sets the output to time

  series format or not

  - **forecast_nsteps** [integer|{12}]: number of forecasting steps
  - **forecast_start_date [char|numeric|serial date]: date when the** forecasts start (end of history + 1)
  - **forecast_conditional_hypothesis [{jma}|ncp|nas]: in dsge models in** which agents have information beyond the current period, this option determines the number of periods of shocks need to match the restrictions: - Hypothesis **jma** assumes that irrespective of how

    many periods of conditioning information are remaining, agents always receive information on the same number of shocks.

    * **Hypothesis ncp assumes there are as many shocks periods as** the number of the number of conditioning periods
    * **Hypothesis nas assumes there are as many shocks periods as** the number of anticipated steps

7.6.3 Outputs

- **cond_fkst_db** [struct|matrix]: depending on the value of **forecast_to_time_series** the returned output is a structure with time series or a cell containing a matrix and the information to reconstruct the time series.

7.6.4 More About

- the historical information as well as the conditioning information come from the same database. The time series must be organized such that for each series, the first page represents the actual data and all subsequent pages represent conditional information. If a particular condition is "nan", that location is not constrained
- Conditional forecasting for nonlinear models is also supported. However, the solving of the implied nonlinear problem may fail if the model displays instability
- Both HARD CONDITIONS and SOFT CONDITIONS are implemented but the latter are currently disabled in expectation of a better user interface.
- The data may also contain time series for a variable with name **regime** in that case, the forecast/simulation paths are computed following the information therein. **regime** must be a member of 1:h, where h is the maximum number of regimes.

7.6.5 Examples

See also: simulate

Help for svar/forecast is inherited from superclass RISE_GENERIC

7.7 get

H1 line

7.7.1 Syntax

7.7.2 Inputs

7.7.3 Outputs

7.7.4 More About

7.7.5 Examples

See also:

Help for svar/get is inherited from superclass RISE_GENERIC

7.8 historical_decomposition

historical_decomposition Computes historical decompositions of a DSGE model

7.8.1 Syntax

```
[Histdec,obj]=history_dec(obj)
[Histdec,obj]=history_dec(obj,varargin)
```

7.8.2 Inputs

- obj : [rise|dsge|rfvar|svar] model(s) for which to compute the decomposition. obj could be a vector of models
- varargin : standard optional inputs **coming in pairs**. Among which: - **histdec_start_date** : [char|numeric|{''}] : date at which the

  decomposition starts. If empty, the decomposition starts at he beginning of the history of the dataset

7.8.3 Outputs

- Histdec : [struct|cell array] structure or cell array of structures with the decompositions in each model. The decompositions are given in terms of: - the exogenous variables - **InitialConditions** : the effect of initial conditions - **risk** : measure of the effect of non-certainty equivalence - **switch** : the effect of switching (which is also a shock!!!) - **steady_state** : the contribution of the steady state

7.8.4 Remarks

- the elements that do not contribute to any of the variables are automatically discarded.
- **N.B** : a switching model is inherently nonlinear and so, strictly speaking, the type of decomposition we do for linear/linearized constant-parameter models is not feasible. RISE takes an approximation in which the variables, shocks and states matrices across states are averaged. The averaging weights are the smoothed probabilities.

7.8.5 Examples

See also:

Help for svar/historical_decomposition is inherited from superclass RISE_GENERIC

7.9 irf

irf - computes impulse responses for a RISE model

7.9.1 Syntax

```
myirfs=irf(obj)

myirfs=irf(obj,varargin)
```

7.9.2 Inputs

- **obj** [rise|dsge|rfvar|svar]: single or vector of RISE models
- **varargin** : optional options coming in pairs. The notable ones that will influence the behavior of the impulse responses are:
- **irf_shock_list** [char|cellstr|{''}]: list of shocks for which we want to compute impulse responses
- **irf_var_list** [char|cellstr|{''}]: list of the endogenous variables we want to report
- **irf_periods** [integer|{40}]: length of the irfs
- **irf_shock_sign** [numeric|-1|{1}]: sign or scale of the original impulse. If **irf_shock_sign** >0, we get impulse responses to a positive shock. If **irf_shock_sign** <0, the responses are negative. If If **irf_shock_sign** =0, all the responses are 0.
- **irf_draws** [integer|{50}]: number of draws used in the simulation impulse responses in a nonlinear model. A nonlinear model is defined as a model that satisfies at least one of the following criteria - solved at an order >1 - has more than one regime and option **irf_regime_specific** below is

    set to false
- **irf_type** [{irf}|girf]: type of irfs. If the type is irf, the impulse responses are computed directly exploiting the fact that the model is linear. If the type is girf, the formula for the generalized impulse responses is used: the irf is defined as the expectation of the difference of two simulation paths. In the first path the initial impulse for the shock of interest is nonzero while it is zero for the second path. All other shocks are the same for both paths in a given simulation.
- **irf_regime_specific** [{true}|false]: In a switching model, we may or may not want to compute impulse responses specific to each regime.
- **irf_use_historical_data** [{false}|true]: if true, the data stored in option **simul_historical_data** are used as initial conditions. But the model has to be nonlinear otherwise the initial conditions are set to zero. This option gives the flexibility to set the initial conditions for the impulse responses.
- **irf_to_time_series** [{true}|false]: If true, the output is in the form of time series. Else it is in the form of a cell containing the information needed to reconstruct the time series.

7.9.3 Outputs

- **myirfs** [{struct}|cell]: Impulse response data

7.9.4 More About

- for linear models or models solved up to first order, the initial conditions as well as the steady states are set to 0 in the computation of the impulse responses.
- for nonlinear models, the initial conditions is the ergodic mean

7.9.5 Examples

See also:

Help for svar/irf is inherited from superclass RISE_GENERIC

7.10 isnan

H1 line

7.10.1 Syntax

7.10.2 Inputs

7.10.3 Outputs

7.10.4 More About

7.10.5 Examples

See also:

Help for svar/isnan is inherited from superclass RISE_GENERIC

7.11 load_parameters

H1 line

7.11.1 Syntax

7.11.2 Inputs

7.11.3 Outputs

7.11.4 More About

7.11.5 Examples

See also:

Help for svar/load_parameters is inherited from superclass RISE_GENERIC

7.12 log_marginal_data_density

H1 line

7.12.1 Syntax

7.12.2 Inputs

7.12.3 Outputs

7.12.4 More About

7.12.5 Examples

See also:

Help for svar/log_marginal_data_density is inherited from superclass RISE_GENERIC

7.13 log_posterior_kernel

H1 line

7.13.1 Syntax

7.13.2 Inputs

7.13.3 Outputs

7.13.4 More About

7.13.5 Examples

See also:

Help for svar/log_posterior_kernel is inherited from superclass RISE_GENERIC

7.14 log_prior_density

H1 line

7.14.1 Syntax

7.14.2 Inputs

7.14.3 Outputs

7.14.4 More About

7.14.5 Examples

See also:

Help for svar/log_prior_density is inherited from superclass RISE_GENERIC

7.15 msvar_priors

H1 line

7.15.1 Syntax

7.15.2 Inputs

7.15.3 Outputs

7.15.4 More About

7.15.5 Examples

See also:

7.16 posterior_marginal_and_prior_densities

H1 line

7.16.1 Syntax

7.16.2 Inputs

7.16.3 Outputs

7.16.4 More About

7.16.5 Examples

See also:

Help for svar/posterior_marginal_and_prior_densities is inherited from superclass RISE_GENERIC

7.17 posterior_simulator

H1 line

7.17.1 Syntax

7.17.2 Inputs

7.17.3 Outputs

7.17.4 More About

7.17.5 Examples

See also:

Help for svar/posterior_simulator is inherited from superclass RISE_GENERIC

7.18 print_estimation_results

H1 line

7.18.1 Syntax

7.18.2 Inputs

7.18.3 Outputs

7.18.4 More About

7.18.5 Examples

See also:

Help for svar/print_estimation_results is inherited from superclass RISE_GENERIC

7.19 prior_plots

H1 line

7.19.1 Syntax

7.19.2 Inputs

7.19.3 Outputs

7.19.4 More About

7.19.5 Examples

See also:

Help for svar/prior_plots is inherited from superclass RISE_GENERIC

7.20 report

REPORT assigns the elements of interest to a rise_report.report object

7.20.1 Syntax

::

- REPORT(rise.empty(0)) : displays the default inputs
- REPORT(obj,destination_root,rep_items) : assigns the reported elements in rep_items to destination_root
- REPORT(obj,destination_root,rep_items,varargin) : assigns varargin to obj before doing the rest

7.20.2 Inputs

- obj : [rise|dsge]
- destination_root : [rise_report.report] : handle for the actual report
- rep_items : [char|cellstr] : list of desired items to report. This list can only include : 'endogenous', 'exogenous', 'observables', 'parameters', 'solution', 'estimation', 'estimation_statistics', 'equations', 'code'

7.20.3 Outputs

none

7.20.4 More About

7.20.5 Examples

See also:

Help for svar/report is inherited from superclass RISE_GENERIC

7.21 set

set - sets options for RISE models

7.21.1 Syntax

```
obj=set(obj,varargin)
```

7.21.2 Inputs

- **obj** [rise|dsge|svar|rfvar]: model object
- **varargin** : valid input arguments coming in pairs.

7.21.3 Outputs

- **obj** [rise|dsge|svar|rfvar]: model object

7.21.4 More About

- one can force a new field into the options by prefixing it with a '+' sign. Let's say yourfield is not part of the options and you would like to force it to be in the options because it is going to be used in some function or algorithm down the road. Then you can run m=set(m,'+yourfield',value). then m will be part of the new options.

7.21.5 Examples

See also:

Help for svar/set is inherited from superclass RISE_GENERIC

7.22 set_solution_to_companion

H1 line

7.22.1 Syntax

7.22.2 Inputs

7.22.3 Outputs

7.22.4 More About

7.22.5 Examples

See also:

7.23 simulate

simulate - simulates a RISE model

7.23.1 Syntax

```
[db,states,retcode] = simulate(obj,varargin)
```

7.23.2 Inputs

- **obj** [rfvar|dsge|rise|svar]: model object
- **varargin** : additional arguments including but not restricted to
  - **simul_periods** [integer|{100}]: number of simulation periods
  - **simul_burn** [integer|{100}]: number of burn-in periods
  - **simul_algo [[{mt19937ar}| mcg16807|mlfg6331_64|mrg32k3a|** shr3cong|swb2712]]: matlab's seeding algorithms
  - **simul_seed** [numeric|{0}]: seed of the computations
  - **simul_historical_data [ts|struct|{''}]: historical data from** which the simulations are based. If empty, the simulations start at the steady state.
  - **simul_history_end_date [char|integer|serial date]: last date of** history
  - **simul_regime [integer|vector|{[]}]: regimes for which the model** is simulated
  - **simul_update_shocks_handle [function handle]: we may want to** update the shocks if some condition on the state of the economy is satisfied. For instance, shock monetary policy to keep the interest rate at the floor for an extented period of time if we already are at the ZLB/ZIF. simul_update_shocks_handle takes as inputs the current shocks and the state vector (all the endogenous variables) and returns the updated shocks. But for all this to be put into motion, the user also has to turn on **simul_do_update_shocks** by setting it to true.
  - **simul_do_update_shocks [true|{false}]: update the shocks based on simul_update_shocks_handle** or not.
  - **simul_to_time_series [{true}|false]: if true, the output is a** time series, else a cell array with a matrix and information on elements that help reconstruct the time series.

7.23.3 Outputs

- **db** [struct|cell array]: if **simul_to_time_series** is true, the output is a time series, else a cell array with a matrix and information on elements that help reconstruct the time series.
- **states** [vector]: history of the regimes over the forecast horizon
- **retcode** [integer]: if 0, the simulation went fine. Else something got wrong. In that case one can understand the problem by running decipher(retcode)

7.23.4 More About

- **simul_historical_data** contains the historical data as well as conditional information over the forecast horizon. It may also include as an alternative to **simul_regime**, a time series with name **regime**, which indicates the regimes over the forecast horizon.

7.23.5 Examples

See also:

Help for svar/simulate is inherited from superclass RISE_GENERIC

7.24 simulation_diagnostics

H1 line

7.24.1 Syntax

7.24.2 Inputs

7.24.3 Outputs

7.24.4 More About

7.24.5 Examples

See also:

Help for svar/simulation_diagnostics is inherited from superclass RISE_GENERIC

7.25 solve

H1 line

7.25.1 Syntax

7.25.2 Inputs

7.25.3 Outputs

7.25.4 More About

7.25.5 Examples

See also:

7.26 stoch_simul

H1 line

7.26.1 Syntax

7.26.2 Inputs

7.26.3 Outputs

7.26.4 More About

7.26.5 Examples

See also:

Help for svar/stoch_simul is inherited from superclass RISE_GENERIC

7.27 svar

~~ no help found

7.28 template

~~ no help found

7.29 theoretical_autocorrelations

H1 line

7.29.1 Syntax

7.29.2 Inputs

7.29.3 Outputs

7.29.4 More About

7.29.5 Examples

See also:

Help for svar/theoretical_autocorrelations is inherited from superclass RISE_GENERIC

7.30 theoretical_autocovariances

H1 line

7.30.1 Syntax

7.30.2 Inputs

7.30.3 Outputs

7.30.4 More About

7.30.5 Examples

See also:

Help for svar/theoretical_autocovariances is inherited from superclass RISE_GENERIC

7.31 variance_decomposition

H1 line

7.31.1 Syntax

7.31.2 Inputs

7.31.3 Outputs

7.31.4 More About

7.31.5 Examples

See also:

Help for svar/variance_decomposition is inherited from superclass RISE_GENERIC

CHAPTER EIGHT

TIME SERIES

ts Methods:

acos - H1 line acosh - H1 line acot - H1 line acoth - H1 line aggregate - H1 line allmean - H1 line and - H1 line apply - H1 line asin - H1 line asinh - H1 line atan - H1 line atanh - H1 line automatic_model_selection - H1 line bar - H1 line barh - H1 line boxplot - H1 line bsxfun - H1 line cat - concatenates time series along the specified dimension collect - H1 line corr - H1 line corrcoef - H1 line cos - H1 line cosh - H1 line cot - H1 line coth - H1 line cov - H1 line ctranspose - H1 line cumprod - H1 line cumsum - H1 line decompose_series - H1 line describe - H1 line display - H1 line double - H1 line drop - H1 line dummy - H1 line eq - H1 line exp - H1 line expanding - H1 line fanchart - H1 line ge - H1 line get - H1 line gt - H1 line head - H1 line hist - H1 line horzcat - H1 line hpfilter - H1 line index - H1 line interpolate - H1 line intersect - H1 line isfinite - H1 line isinf - H1 line isnan - H1 line jbtest - H1 line kurtosis - H1 line le - H1 line log - H1 line lt - H1 line max - H1 line mean - H1 line median - H1 line min - H1 line minus - H1 line mode - H1 line mpower - H1 line mrdivide - H1 line mtimes - H1 line nan - H1 line ne - H1 line numel - H1 line ones - overloads ones for ts objects pages2struct - H1 line plot - H1 line plotyy - H1 line plus - H1 line power - H1 line prctile - Percentiles of a time series (ts) quantile - H1 line rand - H1 line randn - H1 line range - H1 line rdivide - H1 line regress - H1 line reset_start_date - H1 line rolling - H1 line sin - H1 line sinh - H1 line skewness - H1 line sort - H1 line spectrum - H1 line std - H1 line step_dummy - H1 line subsasgn - H1 line subsref - H1 line sum - H1 line tail - H1 line times - H1 line transform - H1 line transpose - H1 line ts - Methods: uminus - H1 line values - H1 line var - H1 line zeros - H1 line

ts Properties:

varnames - names of the variables in the database start - time of the time series finish - end time of the time series frequency - of the time series NumberOfObservations - number of observations in the time series NumberOfPages - number of pages (third dimension) of the time series NumberOfVariables - number of variables in the time series

CHAPTER

NINE

MARKOV CHAIN MONTE CARLO FOR BAYESIAN ESTIMATION

9.1 Metropolis Hastings

9.2 Gibbs sampling

9.3 Marginal data density

9.3.1 Laplace approximation

9.3.2 Modified harmonic mean

9.3.3 Waggoner and Zha (2008)

9.3.4 Mueller

9.3.5 Chib and Jeliazkov

CHAPTER

TEN

DERIVATIVE-FREE OPTIMIZATION

- differential evolution
- bee algorithm
- biogeography
- studga
- ants

CHAPTER

ELEVEN

MONTE CARLO FILTERING

11.1 methods

- [addlistener](mcf/addlistener)
- [cdf](mcf/cdf)
- [cdf_plot](mcf/cdf_plot)
- [correlation_patterns_plot](mcf/correlation_patterns_plot)
- [delete](mcf/delete)
- [eq](mcf/eq)
- [findobj](mcf/findobj)
- [findprop](mcf/findprop)
- [ge](mcf/ge)
- [gt](mcf/gt)
- [isvalid](mcf/isvalid)
- [kolmogorov_smirnov_test](mcf/kolmogorov_smirnov_test)
- [le](mcf/le)
- [lt](mcf/lt)
- [mcf](mcf/mcf)
- [ne](mcf/ne)
- [notify](mcf/notify)
- [scatter](mcf/scatter)

11.2 properties

- [lb] -
- [ub] -
- [nsim] -
- [procedure] -

- [parameter_names] -
- [samples] -
- [is_behaved] -
- [nparam] -
- [is_sampled] -
- [check_behavior] -
- [number_of_outputs] -
- [user_outputs] -
- [known_procedures] -

11.3 addlistener

ADDLISTENER Add listener for event. el = ADDLISTENER(hSource, 'Eventname', Callback) creates a listener for the event named Eventname, the source of which is handle object hSource. If hSource is an array of source handles, the listener responds to the named event on any handle in the array. The Callback is a function handle that is invoked when the event is triggered.

el = ADDLISTENER(hSource, PropName, 'Eventname', Callback) adds a listener for a property event. Eventname must be one of the strings 'PreGet', 'PostGet', 'PreSet', and 'PostSet'. PropName must be either a single property name or cell array of property names, or a single meta.property or array of meta.property objects. The properties must belong to the class of hSource. If hSource is scalar, PropName can include dynamic properties.

For all forms, addlistener returns an event.listener. To remove a listener, delete the object returned by addlistener. For example, delete(el) calls the handle class delete method to remove the listener and delete it from the workspace.

See also MCF, NOTIFY, DELETE, EVENT.LISTENER, META.PROPERTY, EVENTS, DYNAMICPROPS

Help for mcf/addlistener is inherited from superclass HANDLE

Reference page in Help browser doc mcf/addlistener

11.4 cdf

~~ no help found

11.5 cdf_plot

~~ no help found

11.6 correlation_patterns_plot

~~ no help found

11.7 delete

DELETE Delete a handle object. The DELETE method deletes a handle object but does not clear the handle from the workspace. A deleted handle is no longer valid.

DELETE(H) deletes the handle object H, where H is a scalar handle.

See also MCF, MCF/ISVALID, CLEAR

Help for mcf/delete is inherited from superclass HANDLE

Reference page in Help browser doc mcf/delete

11.8 eq

== (EQ) Test handle equality. Handles are equal if they are handles for the same object.

H1 == H2 performs element-wise comparisons between handle arrays H1 and H2. H1 and H2 must be of the same dimensions unless one is a scalar. The result is a logical array of the same dimensions, where each element is an element-wise equality result.

If one of H1 or H2 is scalar, scalar expansion is performed and the result will match the dimensions of the array that is not scalar.

TF = EQ(H1, H2) stores the result in a logical array of the same dimensions.

See also MCF, MCF/GE, MCF/GT, MCF/LE, MCF/LT, MCF/NE

Help for mcf/eq is inherited from superclass HANDLE

11.9 findobj

FINDOBJ Find objects matching specified conditions. The FINDOBJ method of the HANDLE class follows the same syntax as the MATLAB FINDOBJ command, except that the first argument must be an array of handles to objects.

HM = FINDOBJ(H, <conditions>) searches the handle object array H and returns an array of handle objects matching the specified conditions. Only the public members of the objects of H are considered when evaluating the conditions.

See also FINDOBJ, MCF

Help for mcf/findobj is inherited from superclass HANDLE

Reference page in Help browser doc mcf/findobj

11.10 findprop

FINDPROP Find property of MATLAB handle object. p = FINDPROP(H,'PROPNAME') finds and returns the META.PROPERTY object associated with property name PROPNAME of scalar handle object H. PROPNAME must be a string. It can be the name of a property defined by the class of H or a dynamic property added to scalar object H.

If no property named PROPNAME exists for object H, an empty META.PROPERTY array is returned.

See also MCF, MCF/FINDOBJ, DYNAMICPROPS, META.PROPERTY

Help for mcf/findprop is inherited from superclass HANDLE

Reference page in Help browser doc mcf/findprop

11.11 ge

>= (GE) Greater than or equal relation for handles. H1 >= H2 performs element-wise comparisons between handle arrays H1 and H2. H1 and H2 must be of the same dimensions unless one is a scalar. The result is a logical array of the same dimensions, where each element is an element-wise >= result.

If one of H1 or H2 is scalar, scalar expansion is performed and the result will match the dimensions of the array that is not scalar.

TF = GE(H1, H2) stores the result in a logical array of the same dimensions.

See also MCF, MCF/EQ, MCF/GT, MCF/LE, MCF/LT, MCF/NE

Help for mcf/ge is inherited from superclass HANDLE

11.12 gt

> (GT) Greater than relation for handles. H1 > H2 performs element-wise comparisons between handle arrays H1 and H2. H1 and H2 must be of the same dimensions unless one is a scalar. The result is a logical array of the same dimensions, where each element is an element-wise > result.

If one of H1 or H2 is scalar, scalar expansion is performed and the result will match the dimensions of the array that is not scalar.

TF = GT(H1, H2) stores the result in a logical array of the same dimensions.

See also MCF, MCF/EQ, MCF/GE, MCF/LE, MCF/LT, MCF/NE

Help for mcf/gt is inherited from superclass HANDLE

11.13 isvalid

ISVALID Test handle validity. TF = ISVALID(H) performs an element-wise check for validity on the handle elements of H. The result is a logical array of the same dimensions as H, where each element is the element-wise validity result.

A handle is invalid if it has been deleted or if it is an element of a handle array and has not yet been initialized.

See also MCF, MCF/DELETE

Help for mcf/isvalid is inherited from superclass HANDLE

Reference page in Help browser doc mcf/isvalid

11.14 kolmogorov_smirnov_test

tests the equality of two distributions using their CDFs

11.15 le

<= (LE) Less than or equal relation for handles. Handles are equal if they are handles for the same object. All comparisons use a number associated with each handle object. Nothing can be assumed about the result of a handle comparison except that the repeated comparison of two handles in the same MATLAB session will yield the same result. The order of handle values is purely arbitrary and has no connection to the state of the handle objects being compared.

H1 <= H2 performs element-wise comparisons between handle arrays H1 and H2. H1 and H2 must be of the same dimensions unless one is a scalar. The result is a logical array of the same dimensions, where each element is an element-wise >= result.

If one of H1 or H2 is scalar, scalar expansion is performed and the result will match the dimensions of the array that is not scalar.

TF = LE(H1, H2) stores the result in a logical array of the same dimensions.

See also MCF, MCF/EQ, MCF/GE, MCF/GT, MCF/LT, MCF/NE

Help for mcf/le is inherited from superclass HANDLE

11.16 lt

< (LT) Less than relation for handles. H1 < H2 performs element-wise comparisons between handle arrays H1 and H2. H1 and H2 must be of the same dimensions unless one is a scalar. The result is a logical array of the same dimensions, where each element is an element-wise < result.

If one of H1 or H2 is scalar, scalar expansion is performed and the result will match the dimensions of the array that is not scalar.

TF = LT(H1, H2) stores the result in a logical array of the same dimensions.

See also MCF, MCF/EQ, MCF/GE, MCF/GT, MCF/LE, MCF/NE

Help for mcf/lt is inherited from superclass HANDLE

11.17 mcf

~~ no help found

11.18 ne

~= (NE) Not equal relation for handles. Handles are equal if they are handles for the same object and are unequal otherwise.

H1 ~= H2 performs element-wise comparisons between handle arrays H1 and H2. H1 and H2 must be of the same dimensions unless one is a scalar. The result is a logical array of the same dimensions, where each element is an element-wise equality result.

If one of H1 or H2 is scalar, scalar expansion is performed and the result will match the dimensions of the array that is not scalar.

TF = NE(H1, H2) stores the result in a logical array of the same dimensions.

See also MCF, MCF/EQ, MCF/GE, MCF/GT, MCF/LE, MCF/LT

Help for mcf/ne is inherited from superclass HANDLE

11.19 notify

NOTIFY Notify listeners of event. NOTIFY(H,'EVENTNAME') notifies listeners added to the event named EVENTNAME on handle object array H that the event is taking place. H is the array of handles to objects triggering the event, and EVENTNAME must be a string.

NOTIFY(H,'EVENTNAME',DATA) provides a way of encapsulating information about an event which can then be accessed by each registered listener. DATA must belong to the EVENT.EVENTDATA class.

See also MCF, MCF/ADDLISTENER, EVENT.EVENTDATA, EVENTS

Help for mcf/notify is inherited from superclass HANDLE

Reference page in Help browser doc mcf/notify

11.20 scatter

~~ no help found

CHAPTER

TWELVE

HIGH DIMENSIONAL MODEL REPRESENTATION

12.1 methods

- [estimate](hdmr/estimate)
- [first_order_effect](hdmr/first_order_effect)
- [hdmr](hdmr/hdmr)
- [metamodel](hdmr/metamodel)
- [plot_fit](hdmr/plot_fit)
- [polynomial_evaluation](hdmr/polynomial_evaluation)
- [polynomial_integration](hdmr/polynomial_integration)
- [polynomial_multiplication](hdmr/polynomial_multiplication)

12.2 properties

- [N] -
- [Nobs] -
- [n] -
- [output_nbr] -
- [theta] -
- [theta_low] -
- [theta_high] -
- [g] -
- [x] -
- [expansion_order] -
- [pol_max_order] -
- [poly_coefs] -

- [Indices] -
- [coefficients] -
- [aggregate] -
- [f0] -
- [D] -
- [sample_percentage] -
- [optimal] -
- [param_names] -

12.3 estimate

~~ no help found

12.4 first_order_effect

~~ no help found

12.5 hdmr

~~ no help found

12.6 metamodel

~~ no help found

12.7 plot_fit

~~ no help found

12.8 polynomial_evaluation

later on, the function that normalizes could come in here so that the normalization is done according to the hdmr_type of polynomial chosen.

12.9 polynomial_integration

polynomial is of the form a0+a1*x+...+ar*x^r the integral is then a0*x+a1/2*x^2+...+ar/(r+1)*x^(r+1)

12.10 polynomial_multiplication

each polynomial is of the form a0+a1*x+...+ar*x^r

CHAPTER

THIRTEEN

CONTRIBUTING TO RISE

13.1 contributing new code

13.2 contributing by helping maintain existing code

13.3 other ways to contribute

13.4 recommended development setup

13.5 RISE structure

13.6 useful links, FAQ, checklist

CHAPTER

FOURTEEN

ACKNOWLEDGEMENTS

Many people have, oftentimes unknowingly, provided help in the form of reporting bugs, making suggestions, asking challenging questions, etc. I would like to single out a few of them but the list is far from exhaustive:

- Dan Waggoner
- Doug Laxton
- Eric Leeper
- Jesper Linde
- Jim Nason
- Kjetil Olsen
- Kostas Theodoridis
- Leif Brubakk
- Marco Ratto
- Michel Juillard
- Pablo Winnant (dolo)
- Pelin Ilbas
- Raf Wouters
- Tao Zha

CHAPTER

FIFTEEN

BIBLIOGRAPHY

CHAPTER

SIXTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*