Package ‘dynr’

November 22, 2016

Type Package

Title Parse a Dynare model an generate R code

Version 0.1.0

Author Who wrote it

Maintainer Who to complain to <yourfault@somewhere.net>

Description A Dynare parser.

SystemRequirements GNU make

License GPL (>= 2)

LazyData TRUE

RoxygenNote 5.0.1

LinkingTo Rcpp, BH

Imports Rcpp, nleqslv, Matrix, methods, compiler, R6, geigen

Depends regts

Suggests testthat,
knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

compile_model 2
DynMod 2

Index **5**

compile_model *Compile a Dynare model and return a* DynMod

Description

Compile a Dynare model and return a DynMod

Usage

```
compile_model(mod_file, bytecode = TRUE)
```

Arguments

mod_file	the name of the model file (including extension .mod)
bytecode	If TRUE, then the functions used to calculate the residuals and jacobian are compiled.

Value

an DynMod object

DynMod *An R6 class for a Dynare model*

Description

An R6 class for a Dynare model

Usage

```
DynMod
```

Format

R6Class object.

Value

Object of R6Class containing a macro-economic model,

Methods

`get_endo_names()` Returns the names of the endogenous variables.

`get_exo_names()` Returns the names of the exogenous variables.

`get_param_names()` Returns the names of the parameters.

`set_params()` Sets the parameters of the model.

`get_params()` Returns the parameters of the model.

`set_static_exos(exos)` Sets the static values of the exogenous variables. These values are used to compute the steady state.

`get_static_exos()` Returns the static values of the exogenous variables

`set_static_endos(endos)` Sets the static values of the endos variables. These values are used to compute the steady state.

`get_static_endos()` Returns the static values of the endogenous variables, i.e. the values that are supposed to be the steady state values. Function `solve_steady` can be used to compute them. After compiling the model, the static endos are initialized with zeros and the values in the `initval` block in the `mod` file. There is no setter for the static values: you can only modify them by calling function `solve_steady`

`set_period(period)` Sets the model period. `period` is a `regperiod_range` object or an object that can be coerced to a `regperiod_range`. The model period is the longest period for which the model may be solved. This method also allocates storage for all model timeseries. Model timeseries are available for the so called 'model data period', which is the model period extended with a lag and lead period. This method also initialises all model timeseries with static values of the exogenous and endogenous model variables.

`get_period()` Returns the model period

`get_data_period()` Returns the data period, i.e. the model period extended with the lag end lead period

`get_lag_period()` Returns the lag period

`get_lead_period()` Returns the lead period

`set_exo_values(value, names = NULL, period = self$get_data_period())` Sets the value(s) of one more exogenous variables. `value` can be any R object that can be coerced to a numeric. `period` is the period for which endogenous variable is modified. If argument `period` is missing the exo period is used.

`set_exo_data(data, update_mode = c("update", "updval"))` Sets the values of the exogenous variables. `data` is a `regts` or `ts` with column names. If `update_mode` is `"updval"`, then the values are only replaced by non NA values in `data`

`get_exo_data(names, period = self$get_data_period()` Returns the exogenous data

`set_endo_values(value, names = NULL, period = self$get_data_period())` Sets the value(s) of one more endogenous variables. `value` can be any R object that can be coerced to a numeric. `period` is the period for which endogenous variable is modified. If argument `period` is missing then the endo period is used.

`set_endo_data(data, update_mode = c("update", "updval"))` Sets the values of the endogenous variables. `data` is a `regts` or `ts` with column names. If `update_mode` is `"updval"`, then the values are only replaced by non NA values in `data` `set_data(data, update_mode = c("update", "updval"))` Sets the values of the all model variables (both endogenous and exogenouys). `data` is a `regts` or `ts` with column names. If `update_mode` is `"updval"`, then the values are only replaced by non NA values in `data`

`get_endo_data(names, period = self$get_data_period()` Returns the endgenous data

`solve_steady(start = self$get_static_endos(), init_data = TRUE, control = NULL)` Solve the steady state of the model. This methods solves the steady state problem. Argument `start` can be used to specify an initial guess for the steady state values. By default, the initial guess is either based on the `initval` block of the mode file or the result of a previous call of `solve_steady`. If `init_data` is true, then the computed steady state values are used to initialise the endogenous model variables `control` is a list of control options passed to `nleqslv`.

`check()` Compute the eigenvalues of the linear system and check if the Blachard and Kahn conditions are satisfied.

`solve(control = list())` Solves the model using a stacked-time Newton method for the whole model period. Argument `control` is a list with solve options (TODO: describe these options somewhere).

`solve_perturbation()` Solves the model using the perturbation theory used in the Dynare function stoch_simul. Only shocks in the first solution period are allowed.

`get_jacob(sparse = TRUE)` Returns the Jacobian for the stacked-time Newton problem either as a sparse matrix (a `Matrix` object) or normal `matrix`.

`get_eigval(` Returns the eigenvalues of the linearized model. computed with functiomn `check()` of `solve_perturbation()`, ordered with increasing absolute value

Index

∗Topic **data**
    DynMod, 2

compile_model, 2

DynMod, *2*, 2

Matrix, *4*
matrix, *4*

nleqslv, *4*

R6Class, *2*
regperiod_range, *3*