Package 'isismdl'

March 8, 2022

Type Package

Title Solve macroeconomic models

Version 2.1.0

Date 2016-02-18

Author Rob van Harrevelt [aut, cre]

Maintainer Rob van Harrevelt <rvanharrevelt@gmail.com>

Description Solve macroeconomic models.

Depends R (>= 3.5.0),
regts

SystemRequirements GNU make

License GPL-3

RoxygenNote 7.1.2

Roxygen list(markdown = TRUE, r6 = FALSE)

Imports methods,
R6,
readr,
tools

Suggests testthat,
knitr,
R.rsp,
rmarkdown

VignetteBuilder knitr,
R.rsp

R topics documented:

`all.equal` *Test if two* `IsisMdl` *objects are (nearly) equal*

Description

`all.equal(x,y)` is a utility to compare R objects x and y testing near equality. If they are different, comparison is still made to some extent, and a report of the differences is returned. Do not use `all.equal` directly in if expressions - use `isTRUE(all.equal(...))`.

Usage

```
## S3 method for class 'IsisMdl'
all.equal(target, current, ...)
```

Arguments

`target`	and `IsisMdl` object
`current`	another `IsisMdl` object, to be compared with `target`
`...`	Arguments passed to the internal call of `all.equal`.

Details

The implementation of `all.equal` for `IsisMdl` objects first serialized the model using the `IsisMdl` method `serialize` and then uses `all.equal` of the base package.

Value

Either `TRUE` or a character vector describing the differences between target and current.

See Also

`all.equal`

Examples

```
mdl <- islm_mdl("2017Q2/2018Q2")
mdl2 <- mdl$copy()
print(all.equal(mdl, mdl2))

# now modify mdl2
mdl2$set_values(600, names = "c")
print(all.equal(mdl, mdl2))
```

change_data-methods *IsisMdl methods: changes the model data or constant adjustments by applying a function.*

Description

This methods of R6 class IsisMdl changes the model data or constant adjustments by applying a function.

Usage

```
mdl$change_data(fun, names, pattern, period = mdl$get_data_period())

mdl$change_ca(fun names, pattern, period = mdl$get_data_period())
```

mdl is an IsisMdl object

Arguments

fun	a function applied each model timeseries or constant adjustment specified with argument names or pattern
names	a character vector with variable names
pattern	a regular expression
period	an period_range object or an object that can be coerced to a period_range
...	arguments passed to fun

Details

The function specified with argument fun should be a function with at least one argument, for example fun = function(x) {x + 0.1}. The first argument (named x in the example) will be the model variable. The function is evaluated for each model variable separately. The values of the model variables for period range period are passed as a normal numeric vector (not a timeseries) to the first argument.

An example may help to clarify this. Consider the following statement

```
mdl$change_data(fun = myfun, names = c("c", "y"),
                      period = "2017q1/2017q2"),
```

where mdl is a DynMdl object and myfun some function whose details are not relevant here. Method change_data evaluates this as

```
data <- mdl$get_data(names = c("c", "y"), period = "2017q1/2017q2")
data[, "c"] <- myfun(as.numeric(data[, "c"]))
data[, "y"] <- myfun(as.numeric(data[, "y"]))
mdl$set_data(data)
```

The function result must be a vector (or timeseries) of length one or with the same length as the number of periods in the period range period.

Methods

changes_data Changes the model data

change_ca Changes the constant adjustments

See Also

get_data-methods, set_data-methods and set_values-methods

Examples

```
mdl <- islm_mdl(period = "2017Q1/2017Q3")

# increase y and yd with 10% for the full data period
mdl$change_data(pattern = "^y.?$", fun = function(x) {x * 1.1})

# increase ms in 2017Q1 and 2017Q2 with 10 and 20, resp.
mdl$change_data(names = "ms", fun = function(x, dx) {x + dx},
                dx = c(10, 20), period = "2017Q1/2017Q2")
print(mdl$get_data())
```

---

clear_fit IsisMdl *method: deletes all fit targets and rms values values*

Description

This methods of R6 class IsisMdl deletes all fit targets and root mean square (rms) error values for the fit procedure.

Usage

```
mdl$clear_fit()
```

mdl is an IsisMdl object

See Also

set_fit, set_fit_values and get_fit.

clear_fix — IsisMdl *method: deletes all fix values*

Description

This methods of R6 class IsisMdl deletes all fix values specified with methods set_fix, set_fix_values and fix_variables

Usage

```
mdl$clear_fix()
```

mdl is an IsisMdl object

See Also

set_fix, set_fix_values, fix_variables and get_fix

convert_mdl_file — *Converts an IsisMdl a model file.*

Description

Converts an IsisMdl a model file.

Usage

```
convert_mdl_file(
  model_file,
  output_file,
  conversion_options = list(),
  parse_options = list()
)
```

Arguments

model_file	The name of the model file. An extension mdl is appended to the specified name if the filename does not already have an extension
output_file	The name of the output file.
conversion_options	conversion options. See section Conversion options.
parse_options	a named list with options passed to the model parser. See section "Parse option" in the description of function isis_mdl.

Conversion options

The following conversion options can be specified with argument `conversion_options`. This argument should be a named list (for example, `list(substitute = TRUE)`).

`substitute` Specify `TRUE` to substitute user functions. The default is `FALSE`.

`make_dynare` Specify `TRUE` to convert the model to a Dynare mod file.The default is `FALSE`.

---

copy — IsisMdl *method: Returns a copy of this* IsisMdl *object*

Description

This method of R6 class `IsisMdl` returns a deep copy of an `IsisMdl` object

Usage

```
mdl$copy()
```

`mdl` is an `IsisMdl` object

Details

`mdl$copy()` is equivalent to `mdl$clone(deep = TRUE)`

Examples

```
mdl <- islm_mdl("2017Q1/2019Q2")
mdl2 <- mdl$copy()
```

---

fill_mdl_data — IsisMdl *method: Calculates missing model data from identities*

Description

This method of R6 class `IsisMdl` attempts to calculate missing data for endogenous variables of a model by solving the identity equations in solution order.

The procedure can be used to fill in data before and beyond the model period (as set by method `set_period` for as many variables as possible.

Usage

```
mdl$fill_mdl_data(period = mdl$get_data_period(),
                  report = c("period", "minimal", "no"))
```

`mdl` is an `IsisMdl` object

Arguments

`period` a `period_range` object

`report` Defines the type of report about the number of replaced missing values. See details.

Details

Argument `report` can be used to specify the type of report about the number of replaced missing values. Specify

`minimal` to get a minimal report. Only the total number of replaced missing values is reported

`period` to get a report per period (default). For each period the number of replaced missing values is reported

`no` to not generate a report

See Also

`run_eqn`.

Examples

```
mdl <- islm_mdl(period = "2017Q1/2018Q4")

mdl$set_values(200, names = "t", period = "2017Q1")
mdl$fill_mdl_data(period = "2017Q1")
print(mdl$get_data(names = "yd"))
```

`fix_variables` `IsisMdl` *method: Fix variables to their current values*

Description

This method of R6 class `IsisMdl` fixes the specified frml variables to their current values in the model data.

Each frml equation has a constant adjustment. If the frml equation is fixed, then the left hand side of the equation (the frml variable) is kept fixed at the current value in the model data, and the constant adjustment is calculated as the difference between the frml variable and the right hand side of the equation.

Usage

```
mdl$fix_variables(names, pattern, period = mdl$get_period())
```

`mdl` is an `IsisMdl` object

Arguments

`pattern` a regular expression specifying the variable names

`names` a character vector with variable names

`period` an `period_range` object or an object that can be coerced to a `period_range`

If neither `names` nor `pattern` has been specified, then all frml variables are fixed.

See Also

`set_fix`, `get_fix` and `clear_fix`.

Examples

```
mdl <- islm_mdl("2015Q2/2016Q3")
mdl$solve()

# fix variable "c" for a specific period:
mdl$fix_variables(names = "c", period = "2015Q3/2015Q4")

# fix all frml variables
mdl$fix_variables(pattern = ".*")
```

---

get_data-methods — `IsisMdl` *methods: Retrieve timeseries from the model data, constant adjustments, fix values or fit targets*

Description

These methods of R6 class `IsisMdl` can be used to retrieve timeseries from the model data, constant adjustments, fix values or fit targets.

Usage

```
mdl$get_data(pattern, names, period = mdl$get_data_period())

mdl$get_ca(pattern, names, period = mdl$get_data_period())

mdl$get_fix()

mdl$get_fit()
```

`mdl` is an `IsisMdl` object

Arguments

`names` a character vector with variable names

`pattern` a regular expression

`period` an `period_range` object or an object that can be coerced to a `period_range`. The frequency of `period` should be equal to or lower than the frequency of the model period. If the frequency is lower, than the period range is converted to the same frequency as the model frequency with function `change_frequency` of package `regts`.

Methods

- `get_mdl_data`: Model data
- `get_ca`: Constant adjustments
- `get_fix_values`: Fix values
- `get_fit_targets`: Fit targets

Examples

```
mdl <- islm_mdl(period = "2016Q1/2017Q4")

print(mdl$get_data())

# print data for 2017Q2 and later
print(mdl$get_data(names = c("g", "y"), period = "2017Q2/"))

# print data for all quarters in 2017 (2017Q1/2017Q4)
print(mdl$get_data(names = c("g", "y"), period = 2017))

print(mdl$get_data(pattern = "^ymdl"))
```

---

get_data_period *IsisMdl method: returns the model data period*

Description

This method of R6 class `IsisMdl` returns the model data period.

Usage

```
mdl$get_data_period()
```

`mdl` is an `IsisMdl` object

See Also

`set_period`

get_endo_names *IsisMdl method: returns the names of the endogenous model variables*

Description

This method of R6 class IsisMdl returns the names of the endogenous model variables. By default, the function returns the names of all active endogenous variables. Argument pattern can be specified to select only variables with names matching a regular expression. Argument type can be specified to select variables with a specific type. The following types are supported

"frml" stochastic variables. Stochastic variables are variables that occur on the left hand side of frml equations

"lags" all endogenous variables with lags

"leads" all endogenous variables with leads

"feedback" all feedback variables

"all" all endogenous variables, the default

If some equation have been deactivated (see set_eq_status), then argument status may be useful. By default, the function only returns the names of the active endogenous variables, i.e. the variables that occur on the left hand side of active equation. This behaviour can be modified by specifying status. The following options for argument status are recognized:

"active" active endogenous variables, the default

"inactive" inactive endogenous variables

"all" all endogenous variables

Usage

```
mdl$get_endo_names(pattern = ".*",
                   type =  c("all", "frml", "lags", "leads", "feedback"),
                   status = c("active", "inactive", "all"))
```

mdl is an IsisMdl object

Arguments

pattern a regular expression specifying variable names

type a character string specifying the variable type. See the description above

status a character string specifying the status of the endogenous variable (inactive or active). See the description above

See Also

get_exo_names and get_var_names

Examples

```
mdl <- islm_mdl()

# get the names of all stochastic variables
mdl$get_endo_names(type = "frml")

# get all variables with names starting with "y":
mdl$get_endo_names(pattern = "^y.*")
```

`get_eq_names` IsisMdl *method: returns the the equation names*

Description

This method of R6 class IsisMdl returns the the equation names. Argument `pattern` can be specified to select only equations with names matching a regular expression. Argument `status` can be specified to select only the active or inactive equations. Possible options for argument `status` are

`"active"` active equations

`"inactive"` inactive equations

`"all"` all equations, the default

Argument `order` specifies the order of the equations returned. The following ordering options are recognized:

`"sorted"` alphabetically ordering

`"solve"` ordered according to the solution sequence

`"natural"` same order as in the `mdl` file

Usage

```
mdl$get_eq_names(pattern = ".*", status =  c("all", "active", "inactive"),
                 order =  c("sorted", "solve", "natural"))
```

`mdl` is an IsisMdl object

Arguments

`pattern` a regular expression specifying equation names

`status` the equation status, see Description

`order` the ordering of the equations (see description)

Examples

```
mdl <- islm_mdl()

# get the names of equations in solution order
mdl$get_eq_names(order = "solve")

# get all equations with names starting with "y":
mdl$get_eq_names(pattern = "^y.*")
```

---

get_exo_names *IsisMdl method: returns the names of the exogenous model variables*

Description

This method of R6 class IsisMdl returns the names of the exogenous model variables, including the left hand side variables of inactive equations (see set_eq_status).

Usage

```
mdl$get_exo_names(pattern = ".*")
```

mdl is an IsisMdl object

Arguments

pattern a regular expression specifying variable names

See Also

get_endo_names and get_var_names

Examples

```
mdl <- islm_mdl()

# get the names of all exogenous model variables
mdl$get_exo_names()

# get all variables with names starting with "g":
mdl$get_exo_names(pattern = "^g.*")
```

get_labels IsisMdl *method: Returns the labels of the model variables.*

Description

This method of R6 class IsisMdl returns the labels of the model variables.

Usage

```
mdl$get_labels()
```

mdl is an IsisMdl object

See Also

set_labels

get_last_solve_period IsisMdl *method: Returns the last solve period*

Description

This method of R6 class IsisMdl returns the last solve period, i.e. the last period for which method solve attempted to find a solution (whether successful or not). The period is returned as a period object. The function returns NULL when method solve has not yet been used for this IsisMdl object.

Usage

```
mdl$get_last_solve_period()
```

mdl is an IsisMdl object

Examples

```
mdl <- islm_mdl(period = "2018q1/2018q4")$solve()
mdl$get_last_solve_period()
```

get_maxlag — IsisMdl *method: returns the maximum lag of the model*

Description

This method of R6 class IsisMdl returns the maximum lag of the model

Usage

```
mdl$get_maxlag()
```

mdl is an IsisMdl object

get_maxlead — IsisMdl *method: returns the maximum lead of the model*

Description

This method of R6 class IsisMdl returns the maximum lead of the model

Usage

```
mdl$get_maxlead()
```

mdl is an IsisMdl object

get_par_names — IsisMdl *method: returns the names of the model variables*

Description

This method of R6 class IsisMdl returns the names of the model parameters

Usage

```
mdl$get_par_names(pattern = ".*")
```

mdl is an IsisMdl object

Arguments

pattern a regular expression specifying parameter names

Examples

```
mdl <- islm_mdl()

# print all model parameter names
print(mdl$get_par_names())

# print names of model parameters with names starting with c
print(mdl$get_par_names(pattern = "^c.*"))
```

---

get_period IsisMdl *method: returns the model period*

Description

This method of R6 class IsisMdl returns the model period.

Usage

```
mdl$get_period()
```

mdl is an IsisMdl object

See Also

set_period

---

get_solve_status IsisMdl *method: Returns the solve status of the last model solve.*

Description

This method of R6 class IsisMdl returns the status of the last model solve as a text string. If the last model solve was successful, it returns the string "OK".

Usage

IsisMdl method:

```
mdl$get_solve_status()
```

mdl is an IsisMdl object

Details

The possible return values are:

- "Method solve has not yet been called"
- "OK"
- "Simulation not possible" (usually this means that exogenous or feedback variables have NA values)
- "Simulation stopped" (it was not possible to find a solution)
- "Initial lags/leads missing/invalid. Simulation not possible"
- "Invalid parameter values detected. Simulation not possible"
- "Fair-Taylor has not converged"
- "Out of memory. Simulation not successful"
- "Unknown problem in solve. Simulation not successful"

See Also

solve

Examples

```
## Not run:
mdl <- islm_mdl(period = "2017Q1/2018Q4")
mdl$set_values(NA, names = "y", period = "2017Q1")
mdl$solve()
if (mdl$get_solve_status() != "OK") {
   stop("Error solving the model. Check the warnings!")
}

## End(Not run)
```

---

get_text *IsisMdl method: Returns the model text file*

Description

This method of R6 class IsisMdl returns the model text, i.e. the contents of the model file passed to function isis_mdl.
In principle, it is possible to remove the original model file after the IsisMdl object has been created and saved to a file with method write_mdl, since the model text used to create the model is stored in this file. However, this is not a good idea if the model contains preprocessor directives (#include or #if). The current version of isismdl does not handle preprocessor directives yet, but in future versions isismdl will store the preprocessed model text (the model text obtained by evaluating the preprocessor directives). *Therefore, we recommend to always keep the original model file*.

Usage

```
mdl$get_text()
```

mdl is an IsisMdl object

Examples

```
mdl <- islm_mdl()
cat(mdl$get_text())
```

---

get_var_names IsisMdl *method: returns the names of the model variables*

Description

This method of R6 class IsisMdl returns the names of the model variables, both exogenous and endogenous.

Argument type can be specified to select variables with a specific type. The following types are supported

"all" All model variables

"lags" Model variables with lags

"leads" Model variables with leads

Usage

```
mdl$get_var_names(pattern = ".*", type = c("all", "lags", "leads"))
```

mdl is an IsisMdl object

Arguments

pattern a regular expression specifying variable names

type a character string specifying the variable type. See the description above

See Also

get_endo_names and get_exo_names

Examples

```
mdl <- islm_mdl()

# get the names of all model variables
mdl$get_var_names()

# get all variables with names starting with "y":
mdl$get_var_names(pattern = "^y")

# get the names of all lagged variables
mdl$get_var_names(type = "lags")
```

`ifn_mdl` *Returns an IFN model This function returns an uninitialized IFN model.*

Description

Returns an IFN model This function returns an uninitialized IFN model.

Usage

```
ifn_mdl()
```

Value

a `IsisMdl` object

Examples

```
mdl <- ifn_mdl()
```

`init_data` `IsisMdl` *method: initializes the model data.*

Description

This method of R6 class `IsisMdl` initializes the model variables and constant adjustments for the whole data period. The model timeseries are set to `NA` and the constant adjustments to zero.

If arguments `data` or `ca` have been specified, then the model variables or constant adjustments are subsequently updated with the timeseries in `data` or `ca`, respectively. Timeseries in `data` or `ca` that are no model variables or constant adjustments are silently skipped.

If the model period has not yet been specified (in function `isis_mdl` or method `set_period`), then this method also sets the model period, the standard period for which the model will be solved. The model period is obtained from the data period by subtracting the lag and lead periods.

Usage

```
mdl$init_data(data_period, data, ca)
```

`mdl` is an `IsisMdl` object

Arguments

`data_period` a `period_range` object, or an object that can be coerced to `period_range`. If not specified then the data period is based on the period range of argument `data` (if this argument has been specified) and the model period.

`data` a `ts` or `regts` object with model variables

`ca` a `ts` or `regts` object with constant adjustments

See Also

`set_period`

Examples

```
mdl <- islm_mdl()
mdl$init_data("2017Q2/2021Q3")
print(mdl)
```

IsisMdl *An R6 class class representing an Isis model*

Description

This class is used to solve a system of non-linear equations with lagged variables. The model equations are specified in a separate text file, the so called model file. Function `isis_mdl` parses the model file and generates an `IsisMdl` object. The vignette "Introduction" gives a detailed description of the usage of `IsisMdl` objects.

Format

`R6Class` object.

Details

The syntax of the model file is the same of the syntax of model file in Isis, and is described in detail in the Isis Reference Manual (a vignette with a detailed model syntax description for package IsisMdl will be available in a future).

The package included a number of example models in directory `models` of the package library. It is also possible to directly create `IsisMdl` objects with functions `islm_mdl` to create an ISLM model and `ifn_mdl` to create another example model, the IFN model. The latter model is a model with leads and can be solved with the Fair-Taylor-method.

Value

Object of R6Class representing an Isis model.

Methods

IsisMdl objects support the following methods. These methods are described in detail in the different subsection of the documentation For example, method solve is described in section solve.

copy Returns a deep copy of the IsisMdl object

get_text Returns the textual representation of the model

get_maxlag Returns the maximum lag

get_maxlead Returns the maximum lead

get_var_names Returns the names of the model variables

get_exo_names Returns the names of the exogenous model variables

get_endo_names Returns the names of the endogenous model variables

get_par_names Returns the names of the model parameters

get_eq_names Returns the names of the equations

init_data Initializes the model data

set_period Sets the model period

get_period Returns the model period

get_data_period Returns the model data period

set_labels Set labels for the model variables

get_labels Returns the labels of the model variables.

set_param Sets the model parameter

set_param_values Sets the value of one or more model parameter

get_param Returns model parameters

set_data Transfer timeseries to the model data

set_ca Transfer timeseries to the constant adjustments

set_fix Transfer timeseries to the fix values

set_fit Transfer timeseries to the fit targets

set_values Sets the values of the model data

set_ca_values Sets the values of the constant adjustments

set_fix_values Sets the fix values

set_fit_values Sets the values of the fit targets

set_fit Transfer timeseries to the fit targets

change_data Change model data by applying a function

change_ca Change the constant adjusmtments by applying a function

get_data Returns the model data

get_ca Returns the constant adjustments

get_fix Returns the fix values

get_fit Returns the fit targets

set_rms Sets one or more the rms values used in the fit procedure

set_rms_values Sets the rms values used in the fit procedure

set_solve_options Sets the solve options

get_solve_options Returns the solve options

set_fit_options Sets the options for the fit procedure

get_fit_options Returns the options for the fit procedure

set_debug_eqn Sets the debug equation option

get_debug_eqn Returns the debug equation option

set_cvgcrit Sets the convergence criterion for selected variables

set_cvgcrit Returns the convergence criterion for selected variables

set_eq_status Sets the equation status "active" or "inactive")

set_ftrelax Sets the Fair-Taylor relaxtion factors

get_ftrelax Returns the Fair-Taylor relaxtion factors

solve Solves the model

get_solve_status Returns the status of the last model solve attempt

fill_mdl_data Calculates missing model data from identities

write_mdl Serializes the model object and writes it to an RDS file

order Orders the equations of the model

See Also

isis_mdl, islm_mdl and ifn_mdl

Examples

```
# create an example ISLM  model
mdl <- islm_mdl()

# prepare input timeseries
r  <- regts(3.35, start = "2015Q1", end = "2016Q3", labels = "interest rate")
y  <- regts(980,  start = "2015Q1", end = "2016Q3", labels = "income")
yd <- regts(790, start = "2015Q1", labels = "disposable income")
g  <- regts(210 * cumprod(rep(1.015, 6)), start = "2015Q2",
            labels = "government spending")
ms <- regts(200 * cumprod(rep(1.015, 6)), start = "2015Q2",
           labels = "money supply")
islm_input <- cbind(r, y, yd, g, ms)
print(islm_input)

# set period and update model timeseries
mdl$set_period("2015Q2/2016Q3")
mdl$set_data(islm_input)
```

```
mdl$set_labels(c(i = "investment", c = "consumption", md = "money demand",
                 t = "tax"))

mdl$solve()
```

`isis_mdl` *Creates an* `IsisMdl` *object from a model file.*

Description

This function creates an `IsisMdl` object. A model as defined on an external ASCII file is parsed, analyzed and converted into an internal code. This internal code is used to evaluate the model equations.

Usage

```
isis_mdl(
  model_file,
  period,
  data,
  ca,
  fix_values,
  parse_options,
  silent = FALSE
)
```

Arguments

`model_file`	The name of the model file. An extension `mdl` is appended to the specified name if the filename does not already have an extension
`period`	a `period_range` object
`data`	the model data as a `regts` object with column names
`ca`	the constant adjustments as a `regts` object with column names
`fix_values`	the fix values as a `regts` object with column names
`parse_options`	a named list with options passed to the model parser. See section "Parse options"
`silent`	A logical (default `FALSE`). If `TRUE`, then output of the model parser is suppressed.

Details

The file containing the model must have an extension `mdl`.

Some technical information about the model and a cross reference of the model is written to an external file with extension `mrf`. For each variable its maximum lag and lead are given and a list of equations (by name) in which it occurs.

The parser also orders the equations of the model into three separate blocks.

- the *pre-recursive* block containing equations which can be solved recursively from exogenous and lagged variables only.
- the *simultaneous* block containing all equations with interdependent endogenous variables.
- the *post-recursive* block containing equations which can be solved recursively once the two previous blocks have been solved.

The ordering process also provides a list of so-called feedback variables, i.e. variables whose value must be assumed known to make the *simultaneous* block recursive. Initial guesses for these variables must be provided in order to solve a model. If a model has no feedback variables, it is a recursive model (it can be solved in one pass through the equations).

If the parser encounters errors in the model, these are written to a file with an extension `err`. All generated files have the same base name as the model file.

Parse options

The following parse options can be specified with argument `parse_options`. This argument should be a named list

`"flags"` This flag is used for conditional compilation. Consult Section 3.11.2 "Conditional compilation" in the Isis reference manual for more information about conditional compilation

`"include_dirs"` Add directory `include_dir` to the list of directories to be searched for include files. In the model file, the `\#include` directive (see Section 3.11.1 "File inclusion" in the Isis Reference Manual) is used to include another file in the model. The specified name of the include file can be a relative or absolute path. If the path is relative, then the model searches for the include file. It first searches in the same directory where the source file is located. If not found there, then the compiler searches in the directories specified with argument `include_dir`, in the order that the directories have been specified. If the include file is still not found, the parser searches in the current directory

`"gen_dep_file"` Specify `TRUE` to generate a file with dependency information. The default is `FALSE`. The dependency information is written to an external file with extension `dep`. The file gives for each equation a list of the variables that occur in the right hand side of the equation with lags and leads included. This file may be useful for model analysis with other software. Currently this option cannot be used for models with user functions.

See Also

`IsisMdl`, `islm_mdl` and `ifn_mdl`

Examples

```
# copy the islm.mdl file in the directory models of the package
# directory to the current directory
mdl_file <- system.file("models", "islm.mdl", package = "isismdl")
file.copy(mdl_file, "islm.mdl")

mdl <- isis_mdl("islm.mdl")

# an example with parse option "include_dirs":
```

```
mdl <- isis_mdl("islm.mdl", parse_options = list(include_dirs = "mdlincl"))
```

`islm_mdl` *Returns an example ISLM model*

Description

This function returns an example ISLM model, If argument `period` has been specified, then this function also generates some example data for the feedback variables, lags and exogenous variables. The model returned is ready to be solved.

Usage

```
islm_mdl(period = NULL)
```

Arguments

`period`	the model period for the ISLM model

Value

a `IsisMdl` object

Examples

```
mdl <- islm_mdl()
```

`order` `IsisMdl` *method: orders the equations of a model*

Description

This method of R6 class `IsisMdl` orders the equations of a model. This can be useful after (de)activation equations. By specifying argument `orfnam` it is also possible to write ordering information to a file.

Usage

```
mdl$order(orfnam, silent = FALSE)
```

`mdl` is an `IsisMdl` object

Arguments

`orfnam` Name of file on which to print ordering information. If no output file is specified no ordering information will be written

`silent` A logical (default `FALSE`). If `TRUE`, then output is suppressed.

`read_mdl` *Reads an* `IsisMdl` *object from a file*

Description

This function reads a model from a file that has been written by `IsisMdl` method `write_mdl`

Usage

```
read_mdl(file, silent = FALSE)
```

Arguments

`file`	filename (typically with extension `.ismdl`)
`silent`	A logical (default `FALSE`). If `TRUE`, then output is suppressed.

Details

`read_mdl` employs the serialization interface provided by base R function `readRDS`.

Value

an `IsisMdl` object

See Also

`write_mdl`

Examples

```
mdl <- islm_mdl("2017Q1/2019Q2")
mdl$write_mdl("islm_mdl.ismdl")
mdl2 <- read_mdl("islm_mdl.ismdl")
```

run_eqn IsisMdl *method: runs model equations*

Description

This method of R6 class IsisMdl runs specific equations of the model separately for the specified period range. The right-hand sides of the equations are evaluated and used to update the values of the corresponding left-hand side variables in the model data.

If the equation is a stochastic equation (a frml equation) and the corresponding endogenous variable has been fixed, then the constant adjustment of the equation will be calculated such that the result of the equation equals the predetermined required value for the left-hand side.

The names of the equations to be run can be specified with argument names or pattern. These arguments cannot be specified both. If neither argument pattern nor names has been specified, then all active model equations are run.

Usage

```
mdl$run_eqn(pattern, names, period = mdl$get_data_period(),
            solve_order, forwards = TRUE, update_mode = c("upd", "updval"),
            by_period = FALSE)
```

mdl is an IsisMdl object

Arguments

pattern a regular expression. Equations with names matching the regular expression are run.

names a character vector with equation names

period a period_range object or a single period specifying the period range. By default the equation are run for the whole data period.

solve_order A logical: should the specified equations be run in solve order? The default value depends on whether argument names has been specified: FALSE if names has been specified and otherwise TRUE. See Section Equation Order.

forwards A logical indicating whether the equations are evaluated forwards or backwards in time. See Section Forwards and Backwards.

update_mode This argument specifies whether the model data should be updated with the result of running an equation if the result is an invalid number (NA). If update_mode = "upd" (the default), the model data is always updated with the result. If update_mode = "updval", the model data is only updated if the result is not NA.

by_period A logical (default FALSE). If TRUE, and if forwards is TRUE, all equations are first evaluated at the first period, then all equations at the second period, and so on. If by_period is FALSE, the first equation is first run for all periods (starting at the first period, then the second period etc.), then the second equation is solved for all periods, and so on. If by_period is TRUE and if forwards is FALSE, all equations are first evaluated at the last period, then all equations at the last but one period, and so on. If by_period is FALSE, the first equation is first run for all

periods (starting at the last period, then the last but one period etc.), then the second equation is solved for all periods, and so on.

Only one of the two arguments `pattern` and `names` can be specified.

Equation order

If argument `solve_order = TRUE`, the specified equations are run in solve order, i.e. the order used when solving the model. If `solve_order = FALSE`, the order depends on whether argument `names` is specified:

- If argument `names` has been specified, the equations are run in the same order as the specified names.
- Otherwise the equations are run using the 'natural order', i.e. the order of the equations as defined in the model file. The default of argument `solve_order` is `FALSE` if `names` has been specified and `TRUE` in other cases.

Forwards and Backwards

By default, the equations are run forwards in time: the equations are first evaluated at the first period of the specified period range, then at the second period, and so on. If argument `forwards = FALSE`, the equations are run backwards: first at the last period of the specified period range, then at the last but one period, and so on.

See Also

`solve` and `fill_mdl_data`.

Examples

```
mdl <- islm_mdl("2017Q1/2019Q3")
mdl$run_eqn(names = c("c", "t"))

# run all equations with names starting with y
mdl$run_eqn(pattern = "^y")

# run all model equations in the order of the equations as specified
# in the mdl file
mdl$run_eqn(solve_order = FALSE)

# emulate a single pass through the model
# note that we use by_period = TRUE
mdl$run_eqn(period = mdl$get_period(), by_period = TRUE)
```

serialize — *Serializes the model to an* serialized_isismdl *S3 class*

Description

This method of R6 class IsisMdl serializes the model object and returns an serialized_isismdl object, an S3 object that contains all the information about the model. The serialized model can be used to create a new IsisMdl object with the command IsisMdl$new(serialized_mdl)

Usage

```
mdl$serialize()
```

mdl is an IsisMdl object

See Also

write_mdl and read_mdl

Examples

```
mdl <- islm_mdl("2017Q1/2019Q2")
serialized_mdl <- mdl$serialize()

# create a new model from the serialized model
mdl2 <- IsisMdl$new(serialized_mdl)
```

set_cvgcrit — IsisMdl *method: Sets the convergence criterion for selected variables.*

Description

This method of R6 class IsisMdl sets the convergence criterion for one or more endogenous model variables. A variable x has converged when two successive values x_2 and x_1 satisfy the following condition

$$|x_2 - x_1| \leq \epsilon \max(1, |x_1|)$$

where ϵ is the convergence criterion for the tested variable.

The default value of ϵ for all variables is the square root of the machine precision (sqrt(.Machine$double.eps), typically about 1.5e-8)

Method get_cvgcrit() returns the convergence criteria for all model variables

Usage

```
mdl$set_cvgcrit(value, pattern, names)

mdl$get_cvgcrit()
```

mdl is an IsisMdl object

Arguments

value	convergence criterion. This must be a small positive number
pattern	a regular expression specifying the variable names
names	a character vector with variable names

If neither pattern nor names have been specified, then the convergence criterion of all endogenous variables will be set to the specified value.

Examples

```
mdl <- islm_mdl()

# set convergence criterion for variables "c" and "i":
mdl$set_cvgcrit(1e-4, names = c("c", "i"))

# set convergence criterion for variables "y" and "yd":
mdl$set_cvgcrit(1e-4, pattern = "^y*")

print(mdl$get_cvgcrit())
```

---

set_data-methods — IsisMdl *methods: transfers data from a timeseries object to the model data, constant adjustments, fix values or fit targets.*

Description

These methods of R6 class IsisMdl Transfers data from a timeseries object to the model data, constant adjustments, fix values or fit targets.

Usage

```
mdl$set_data(data, names = colnames(data), upd_mode = c("upd", "updval"),
             fun, name_err = c("silent", "warn", "stop"))

mdl$set_ca(data, names = colnames(data), upd_mode = c("upd", "updval"),
           fun, name_err = c("silent", "warn", "stop"))

mdl$set_fix(data, names = colnames(data), upd_mode = c("upd", "updval"),
            name_err = c("silent", "warn", "stop"))
```

```
mdl$set_fit(data, names = colnames(data), upd_mode = c("upd", "updval"),
            name_err = c("silent", "warn", "stop"))
```

mdl is an IsisMdl object

Arguments

data a ts or regts timeseries object

names a character vector with variable names, with the same length as the number of timeseries in data. Defaults to the column names of data. If data does not have column names, then argument names is mandatory

upd_mode the update mode, a character string specifying how the timeseries are updated: "upd" (standard update, default) or "updval" (update only with valid numbers). See details.

fun a function used to update the model data. This should be a function with two arguments. The original model data is passed to the first argument of the function and data to the second argument. See the examples.

name_err A character that specifies the action that should be taken when a name is not the name model variable of the appropriate type (for set_fit, the variable must be endogenous, for set_fix a frml variable). For "silent" (the default), the variable is silently skipped, for "warn" a warning is given and for "stop" an error is issued.

Methods

set_data Sets model data. If data has labels, then set_data will also update the labels of the corresponding model variables

set_ca Set constant adjustments, i.e. the residuals of behavioral (frml) equations

set_fix Set fix values for frml variables (model variables that occur at the left hand side of a frml equation). The frml variable is kept fixed at the specified value, and the constant adjustment of the frml equation is computed at the difference between the frml variable and the right hand side of the equation. A fix value of NA implies that the corresponding variable is *not* fixed. set_fix also updates the model data with all non-NA values in data.

set_fit Set fit targets for the fit procedure. A fit target value of NA implies that the corresponding variable is no fit target

Details

Method set_data transfers data from a timeseries object to the model data. If data is a multivariate timeseries object, then each column is used to update the model variable with the same name as the column name. If data does not have column names, or if the column names do not correspond to the model variable names, then argument names should be specified.

By default, all values in data are used to update the corresponding model variable. Sometimes it is desirable to skip the NA values in data. This can be achieved by selecting "updval" for argument upd_mode. Other non finite numbers (NaN, Inf, and -Inf) are also disregarded for this update mode.

set_ca, set_fix and set_fit and set_data work similarly.

See Also

get_data-methods, set_values-methods, change_data-methods, fix_variables, clear_fix and clear_fit.

Examples

```
mdl <- islm_mdl(period = "2017Q1/2017Q3")

# create a multivariate regts object for exogenous variables g and md
exo <- regts(matrix(c(200, 210, 220, 250, 260, 270), ncol = 2),
             start = "2017Q1", names = c("g", "ms"))

# set and print data
mdl$set_data(exo)
print(mdl$get_data())

# create a univariate regts object for exogenous variable ms,
# with a missing value in 2017Q2
ms <- regts(c(255, NA, 273), start = "2017Q1")

# update with update mode updval (ignore NA)
# note that here we have to specify argument names,
# because ms does not have column names
mdl$set_data(ms, names = "ms", upd_mode = "updval")
print(mdl$get_data())

# in the next example, we use argument fun to apply an additive shock to the
# exogenous variables g and ms.
shock <- regts(matrix(c(-5, -10, -15, 3 , 6, 6), ncol = 2),
             start = "2017Q1", names = c("g", "ms"))
mdl$set_data(shock, fun = function(x1, x2) {x1 + x2})

# the statement above can be more concisely written as
mdl$set_data(shock, fun = `+`)
#`+` is a primitive function that adds its two arguments.


# fix c in 2017Q1/2017q2 and i in 2017q1 to specific values
c <- regts(250, period = "2017q1/2017q2")
i <- regts(175, period = "2017q1")
fix_data <- cbind(c, i)
mdl$set_fix(fix_data)
```

---

set_debug_eqn IsisMdl *method: R Sets the debug equation option*

Description

This method of R6 class IsisMdl sets the debug equation option (TRUE or FALSE)

Method get_debug_eqn() returns the debug equation option.

Usage

```
mdl$set_debug_eqn(value)

mdl$get_debug_eqn()
```

mdl is an IsisMdl object

Arguments

value A logical. If TRUE, then equation debugging is turned on. The default is FALSE

Details

When a model cannot be solved this may be caused by errors in the model equations or even errors in the initial data. If debug mode is set to on, Isis will print messages in the output file whenever it encounters numerical problems during calculation of an equation.

See Also

set_solve_options

Examples

```
mdl <- islm_mdl()
mdl$set_debug_eqn(TRUE)

print(mdl$get_debug_eqn())
```

---

set_eq_status IsisMdl *method: activates or de-activates one or more equations.*

Description

This method of R6 class IsisMdl can be used to set the equation status (active or inactive) of one or more equations.

This procedure is used to activate or deactivate a specified set of equations. After compiling a model, all equations are active. Sometimes however it can be necessary to temporarily exclude an equation from the model and the solution process without actually removing it.

Deactivating an equation implies that the left-hand side variable becomes an exogenous variable. As long as an equation is inactive, the corresponding left-hand side variable and any constant adjustment (for frml equations) will remain *unchanged* in the model workspace.

However the methods set_data, set_ca, get_data and get_ca will still transfer data to and from the model workspace.

A deactivated equation can also be reactivated. It will again participate in the solution process and its left-hand side variable will be treated as endogenous.

Since deactivating effectively changes the structure of the model, it may be necessary to compute a new ordering of the model. This is not done automatically. Use method order.

If the left-hand side variable of a deactivated equation appears as lead in the model, that lead will temporarily be marked as an exogenous lead. However, if a lead of another endogenous variable occurs only in the deactivated equation that particular lead will *not* be registered as exogenous. The model will still be regarded as containing endogenous leads and therefore the default solution mode will be ratex, i.e. the Fair-Taylor method will be used for solving the model.

Usage

```
mdl$set_eq_status(status =  c("active", "inactive"), pattern, names)
```

mdl is an IsisMdl object

Arguments

status a character string specifying the equation status ("active" or code"inactive")

pattern a regular expression specifying the names of the equations

names a character vector with the names of the equations

If neither pattern nor status have been specified, then all equations will be activated or deactivated.

See Also

get_eq_names and order

Examples

```
mdl <- islm_mdl()

# deactivate equation "c" and "i"
mdl$set_eq_status("inactive", names = c("c", "i"))

# deactivate all equations starting with "y" ("y" and "yd")
mdl$set_eq_status("inactive", pattern = "^y*")

# print all deactivated equations
print(mdl$get_eq_names(status = "inactive"))
```

set_fit_options IsisMdl *method: Sets the options for the fit procedure.*

Description

This method of R6 class IsisMdl can be used to set one or more options for the fit procedure. These options will be stored in the IsisMdl object.

Method get_fit_options returns the solve options as a named list

Usage

```
mdl$set_fit_options(maxiter, cvgabs, mkdcrt, cvgrel, zero_ca, warn_ca,
                    accurate_jac, zealous, scale_method,
                    warn_zero_row, warn_zero_col,
                    chkjac, report, dbgopt, svdtest_tol)

mdl$get_fit_options()
```

mdl is an IsisMdl object

Arguments

All arguments below expect a numerical value unless mentioned otherwise.

maxiter The maximum number of iterations (default 5)

cvgabs Criterion for absolute convergence. When the largest scaled discrepancy of the fit target values is less than cvgabs, the fit procedure has converged. The default value is 100 times the square root of the machine precision (100 * sqrt(.Machine$double.eps)), which is typically 1.5e-6.

mkdcrt Criterion for calculating a new fit Jacobian. When the ratio of two successive largest scaled discrepancies of the fit target values is larger than mkdcrt a new fit Jacobian will be calculated in the next iteration. Any value specified must lie between 0.05 and 0.95. The default value is 0.5.

cvgrel Criterion for accepting the result of a fit iteration (default 0.95). When the ratio of two successive largest scaled discrepancies of the fit target values is larger than cvgrel, then the result of the iteration is rejected. If the iteration employed an old fit Jacobian (i.e. a Jacobian computed in an earlier iteration), then a second attempt with a new Jacobian is made. If the iteration already used a new Jacobian, then the fit procedure will be terminated.

zero_ca A logical. If TRUE, then the initial values of the constant adjustments used in the fit procedure are initialized to 0. The default is FALSE

warn_ca A logical. If TRUE (default), then warnings are given for possibly too large constant adjustments at the end of the fit procedure for each period.

accurate_jac A logical. If TRUE (default), then the fit Jacobian is calculated accurately, otherwise the Jacobian is calculated approximately. See Details.

`zealous` A logical. If `TRUE` (default), then a zealous version of the fit procedure is used (see section The Zealous and Lazy Fit Method), otherwise a lazy version is used. The recommended option is to use the zealous version, although this may require much more CPU time.

`scale_method` The scaling method for the fit Jacobian. Possible values are `"row"` (row scaling, the default), and `"none"` (no scaling). See Section "Row Scaling".

`warn_zero_row` A logical (default `FALSE`). If `TRUE`, then a warning is issued for each row of the fit Jacobian for which all values are almost equal to zero. A row of the fit Jacobian contains the derivatives of a fit targets with respect to the residuals. A row is considered almost zero if the L1-norm of that row is smaller than a fraction ϵ of the largest L1-norm of the rows. ϵ is the square root of the machine precision (typically `1.5e-8`). If row scaling is applied (see argument `scale_method`), then a row that is almost zero is usually not problematic. However, if all values in a row are *exactly* zero, then the fit procedure is not possible and therefore an message is always issued, even if `warn_zero_row = FALSE`.

`warn_zero_col` A logical (default `FALSE`). IF `TRUE`, then a warning is issued for each column of the fit Jacobian for which all values are almost or exactly equal to zero. A column of the fit Jacobian contains the derivatives of all fit targets with respect to one particular residual. A columns is considered almost zero if the L1-norm of that column is smaller than a fraction ϵ of the largest L1-norm of the columns. ϵ is the square root of the machine precision (typically `1.5e-8`). A column that is (almost) zero is not necessarily problematic, except when the number of non-zero columns is smaller than the number of rows (the number of fit targets). In particular, if the number of columns with a norm exactly equal to zero is larger than the difference between the number of rows and the number of columns, then the fit procedure is not possible. Therefore a message about zero columns is always given in that case.

`chkjac` A logical. If `TRUE` (the default), then the fit method is terminated when the inverse condition of the fit Jacobian is smaller than the square root of the machine precision (typically `1.5e-8`). When a model is badly scaled, the inverse condition number of the Jacobian may become small, which can lead to inaccurate or even unstable solutions. If `FALSE`, the fit procedure is only terminated when the inverse condition is exactly zero.

`report` A character string specifying the the type of report of the fit procedure for each period. Possible values are `"fullrep"` (the default, an iteration report is printed for each period) and `"minimal"` (for a one line summary).

`dbgopt` A character vector specifying one or more debugging options. See section "Debugging Options" below

`svdtest_tol` Singular Value Decomposition (SVD) test tolerance parameter. The default value for argument `svdtest_tol` is `-1`, which implies that the SVD test is never performed. Specify a number between 0 and 1 to enable an SVD analysis, depending on the inverse condition of the Jacobian. See section "SVD Analysis'. If scaling has been applied (see argument `scale_method`), then the SVD analysis is performed for the scaled Jacobian. Sometimes it is easier to interpret the result of the SVD analysis by turning off row scaling. When this option has been specified, a copy of the fit Jacobian is kept in memory, even if the Jacobian is not ill-conditioned. For large models this option should therefore only be used during testing, and should be turned off in production calculations

Details

The purpose of the fit procedure is to adjust a model solution to a partial set of known outcomes for endogenous variables. It determines the minimal norm vector of specified constant adjustments

which ensure that the specified endogenous variables meet the desired outcome (fit targets). It can be used amongst others to update a model forecast given a (small) set of recent observations of endogenous variables. It first solves the model for any period given all data and if fit targets have been specified then proceeds to determine a set of constant adjustments that will ensure that the fit targets are met. There must be at least as many constant adjustments as there are fit targets for the fit procedure to work.

After solving the model in any period for a given set of values of the constant adjustments (residuals), the fit problem can be described as follows. Find a minimum norm vector u such that

$$y = h(u) = w$$

where u is an n-vector of scaled residuals, y is an m-vector of endogenous variables with $n >= m$, h the function $h : R^n \rightarrow R^m$ and w is an m-vector of fit target values. The scaled residuals u_i have been scaled with the root mean square values specified with procedures `set_rms`.

The fit procedure linearizes the relation $y = h(u)$ and determines a minimum norm solution for u to the resulting set of linear equations after setting $y = w$. It uses the QR decomposition for numerical stability.

The fit Jacobian $D_{ij} = \partial h_i / \partial u_j$ is calculated numerically by a first difference approach. The j'th column is calculated by giving residual u_j a small distortion and then solving the model again. For numerical efficiency the model is solved with a *single* iteration by default. This is usually a good approximation. Use argument `accurate_jac = TRUE` for a more accurate calculation of the fit Jacobian. For this option the model is solved until convergence has been reached.

The criterion used for testing for convergence is the largest scaled discrepancy of the fit target values at iteration k defined as

$$F_k = \max_i \{|w_i - y_i| / \max(|w_i|, 1)\}$$

When $F_k \leq \epsilon$ where $0 < \epsilon < 1$, absolute convergence has been achieved. The value of $epsilon$ is specified with argument `cgvabs (` (the default value is 100 times the square root of the machine precision, which is typically `1.5e-6`). If the zealous fit method is used (see Section The Zealous and Lazy Fit Method), we also require for convergence that the relative step size for all variables is smaller than $epsilon$.

Since evaluating the Jacobian of $h(u)$ can be a time-consuming process, the Jacobian of a previous iteration can sometimes be reused for a next iteration. As long as $F_k \leq \delta F_{k-1}$ where $0 < \delta < 0.95$ the current Jacobian will not be recalculated, except when the zealous fit method is used and the the number of residuals is larger than the number of targets, see Section The Zealous and Lazy Fit Method. The default value for δ is 0.5. When $F_k > 0.95 F_{k-1}$ and the current Jacobian is not up-to-date, the residuals will be reset to the values of the previous iteration and the Jacobian will be recalculated. However if the current Jacobian is up-to-date, the process will be stopped with the message `Cannot locate a better point`.

If the zealous fit method is used (see next paragraph), then a new Jacobian is calculated every iteration when the number of residuals is larger than the number of targets ($m > n$).

The Zealous and Lazy Fit Method

There are two implementations of the fit procedure: the lazy and zealous method. The default is the zealous method. For the lazy method the fit iterations is terminated when the largest scaled

discrepancy of the fit target values is less than `cvgabs`. However, the other variables may not be converged yet sufficiently, particularly when the number of residuals is larger than the number of targets ($m > n$). For the zealous fit procedure continues iterating until the relative changes of all variables are less than `cvgabs`. These relative changes are shown in the output as `Delsmx` (maximum step size in an iteration). The zealous fit procedure also uses an accurate calculation of the Jacobian (see general description). If $m > n$ (non-square fit problem), the zealous fit procedure also updates the fit Jacobian every iteration, because for non-square fit problems the results depends on the Jacobian. For the square case $m = n$ this is not necessary because the final results are independent on the Jacobian.

Row scaling

As explained in section Details, the fit Jacobian D_{ij} is a matrix with the derivatives of the fit targets (i) with respect to the scaled residuals (j). If there are large scale differences between the fit targets, additional row scaling may improve the condition number of the fit Jacobian.

The following procedure is used to determine if row scaling is necessary. For each row i, the maximum absolute values R_i is determined. If if the ratio of the largest and smallest value of vector R is larger than 10, then all rows are scaled so that the largest absolute value in each row is 1. If the ratio is smaller than 10, the Jacobian is not scaled.

Row scaling can be turned off by specifying argument `scale_method = "none"`.

Debugging Options

Argument `dbgopt` can be used to specify one or more options for debugging the fit procedure. Possible values are

`prica` print the constant adjustments values and changes at each fit iteration.

`noprica` do not print the constant adjustments values and changes at each fit iteration.

`prijac` print the fit Jacobian every time it is calculated.

`noprijac` do not print the fit Jacobian every time it is calculated.

`supsot` to suppress all output of the normal solution process.

`nosupsot` to not suppress all output of the normal solution process. Output will be a mess if this option is used.

The default debug options are `c("noprica","noprijac","supsot")`

SVD Analysis

If the inverse condition of the fit Jacobian is exactly zero, then it is impossible to solve the equations of the fit procedure, and the fit procedure is terminated. When the inverse condition is small but non-zero, the solution is often inaccurate or even unstable. In some cases the (near) singularity is caused by (almost) zero rows or columns of the fit Jacobian. It is also possible that some rows or columns are linearly dependent. The example below shows a case where the rows are dependent.

The Singular Value Decomposition (SVD) (see the Wikipedia article about SVD (https://en.wikipedia.org/wiki/Singular_value_decomposition) may help to find the linear dependent rows and columns. The SVD analysis can be enabled by specifying argument `svdtest_tol` of `set_fit_options`.

The output of the SVD analysis are the left and right singular vectors of the Jacobian. A left singular vector is a linear combination of the rows of the Jacobian that is almost zero. A right singular vector is a linear combination of the columns that is almost zero. An example for the ISLM model is shown below.

First we create an Isis model and prepare fit data.

```
mdl <- islm_mdl("2020Q1")
y <- regts(985, start = "2020q1")
yd <- regts(800, start = "2020q1")
c <- regts(600, start = "2020q1")
fit <- cbind(y, yd, c)
mdl$set_fit(fit)
mdl$set_rms(c(c = 5.0, i = 21, md = 2))
```

So we have the following fit targets:

```
mdl$get_fit()
```

```
##          c   y  yd
## 2020Q1 600 985 800
```

We specify fit options so that the SVD analysis is performed and the fit Jacobian is printed.

```
mdl$set_fit_options(svdtest_tol = 1e-8, dbgopt = "prijac")
```

Next solve the model. Because y and yd are related according to `y = yd -t` (`t` is also linearly related to `y`, so there is a linear relation between `y` and `yd`), the fit Jacobian contains dependent rows.

```
mdl$solve()
```

```
##
## Model Solve Options
## Solution period               2020Q1/2020Q1
## Simulation mode               dynamic
## Feedback starting values      current period
## Maximum iterations per period 50
## Relaxation minimum            0.500E-01
##            maximum            1.00
##            shrinkage          0.500
## Criteria stepback             1.30
##           matrix              0.900
## Maximum updates Newton matrix per period                10
## Maximum number of line searches with old Jacobian       5
## Criterion for line search decisions etc.                geometric
##
##
##      3 CAs used by Fit:
##      c  i  md
```

```
##    1 steps for Newton matrix at iteration   0 (1/condscal =  1.43E-01)
## Convergence for 2020Q1 in    4 iterations
##  Fiter     Icond       Delwmx       Delsmx Deltyp        Ratio Type Name
##      0                0.00652              w                    Rel  c
##
## Fit jacobian (scaled with rms) in period 2020Q1       at iteration    1
##
##               c            i           md
## y       8.04217     33.77122     -2.14157
## yd      6.27289     26.34155     -1.67042
## c       7.72048     11.42037     -2.05591
##
## Fit Warning - D matrix is ill conditioned.
## Inverse condition=  6.72E-16 < Machine prec**.5=  1.49E-08
## Derivatives of fit targets are dependent or
## for one or more fit targets all derivatives are (almost) zero ...
## Tip: try to use svd analysis and/or fit option 'warn_zero_row' or warn_zero_col'.
## See documentation of method set_fit_options.
##
## *** SVD analysis ***
## ======================
## The purpose of the SVD analysis is to find linearly dependent rows or columns
## of the fit jacobian.
## A left singular vector is a linear combination of the rows of the jacobian
## that is (almost) zero.
## A right singular vector is a linear combination of the columns of the jacobian
## that is (almost) zero.
## Only components >=   0.15E-07  are shown.
##
## Left Singular vectors:
## ----------------------
## Singular value   0.73E-15
##                                y -0.62
##                               yd  0.79
##
## Right Singular vectors:
## -----------------------
## Singular value   0.73E-15
##                                c  0.26
##                               md  0.97
##
##
## The singularity may also be caused by (almost) zero rows or columns.
## Therefore we print the norm of "problem rows" and "problem columns"
## (rows and columns with significant components in left and right singular vectors resp.)
##
## Problem rows:
## -------------
```

```
##                          Variable     L1 norm of row
##                                 y   44.
##                                yd   34.
##
## Problem columns:
## ----------------
##                          Variable  L1 norm of column
##                                 c   22.
##                                md   5.9
##
## *** END SVD ANALYSIS ***
##


## Warning in mdl$solve(): Simulation not possible


## Total number of iterations   14
## Solve model used     0.01 CPU secs
##
```

Examples

```
mdl <- islm_mdl("2020Q1")

# print constant adjustment and Jacobian  for each fit iteration
mdl$set_fit_options(zealous = TRUE, dbgopt = c("prica", "prijac"))
```

`set_ftrelax` IsisMdl *method: Sets the Fair-Taylor relaxation factors*

Description

This method of R6 class IsisMdl sets the Fair-Taylor relaxation factors for the endogenous leads.

Method `get_ftrelax()` returns the Fair-Taylor relaxation factors for all endogenous leads.

Usage

```
mdl$set_ftrelax(value, pattern, names)

mdl$get_ftrelax()
```

`mdl` is an IsisMdl object

Arguments

`value` Fair-Taylor relaxation number. This must be a positive number or `NA` to disable any previously set value. The default value for all endogenous leads is `NA`, which means that the general uniform Fair-Taylor relaxation (solve option `ftrelax`, see `set_solve_options`) will be applied

`pattern` a regular expression specifying the variable names

`names` a character vector with variable names

If neither `pattern` nor `names` have been specified, then the Fair-Taylor relaxation factors of all variables with endogenous leads will be set to the specified values.

Examples

```
mdl <- ifn_mdl()

# set Fair-Taylor relaxation factor all all variables with names of length 2
# to 0.5:
mdl$set_ftrelax(0.5, pattern = "^..$")

# set Fair-Taylor relaxation factor for variable "lambda":
mdl$set_ftrelax(0.5, names = "lambda")

print(mdl$get_ftrelax())
```

---

set_labels *IsisMdl method: Sets labels for the model variables.*

Description

This method of R6 class `IsisMdl` sets labels for the model variables.

Usage

```
mdl$set_labels(labels)
```

`mdl` is an `IsisMdl` object

Arguments

`labels` a named character vector. The names are the names of the model variables

Examples

```
mdl <- islm_mdl()
mdl$set_labels(c(c = "Consumption", i = "investments"))
```

set_param IsisMdl *method: Set the values of model parameters*

Description

Method set_param and set_param_values of R6 class IsisMdl can be used to set the values of model parameters. A parameter may have more than one element. The parameter value is a numeric vector of the appropriate length.

Method set_param can be used to specify individual parameters, while set_param_values is a convenient method to give more than one parameter the same value.

Method get_param returns a list with the values of the parameters.

Usage

```
mdl$set_param(p, name_err = c("warn", "stop", "silent"))

mdl$set_param_values(value, names, pattern)

mdl$get_param(pattern, names)
```

mdl is an IsisMdl object

Arguments

p a named character vector or list. The names are the names of the parameter names. p should be a list if one or more of the specified parameters are vector parameters, i.e. parameters with a length greater than 1. In that case, the list elements are numeric vectors with a length equal to the length of the corresponding parameter.

name_err A character that specifies the action that should be taken when a name is not the name of a parameter. For "warn" (the default), a warning is given, for "stop" an error is issued. For "silent", the variable is silently skipped.

value A numeric vector of the appropriate length. All parameters specified with argument names and pattern must have the same length as argument value.

names A character vector specifying the names of the parameters.

pattern A regular expression. The action (get or set parameter values) is applied to all parameters with names matching pattern.

If neither names nor pattern has been specified in methods set_param_values or get_param, then the action is applied to all model parameters.

Examples

```
mdl <- islm_mdl()
mdl$set_param(list(i0 = 101))

# give parameters i0, c0, m0, and t0 the value 0
```

```
mdl$set_param_values(0, pattern = ".0")

# print all parameters
mdl$get_param()

# print parameters c0, c1, c2 and c3
print(mdl$get_param(pattern = "^c.*"))
```

---

set_period IsisMdl *method: sets the model period*

Description

This method of R6 class IsisMdl sets the model period. This is the default period used when solving the model.

If the model data has not already been initialized with method init_data, then set_period also initializes the model data. In that case the model data period is set to the specified model period extended with a lag and lead period. Model timeseries are initialized with NA and all constant adjustments with 0.

If the model data has already been initialized with method init_data, then the new model period should be compatible with the model data period. In particular, the new model period extended with a lag and lead period should not contain periods outside the model data period.

Usage

```
mdl$set_period(period)
```

mdl is an IsisMdl object

Arguments

period period_range object, or an object that can be coerced to period_range

Examples

```
mdl <- islm_mdl()
mdl$set_period("2017Q2/2021Q3")
```

set_rms IsisMdl *method: Sets the root mean square errors*

Description

Methods set_rms and set_rms_values of R6 class IsisMdl can be used to set the root mean square (rms) error values used in the fit procedure. Each frml equation has a constant adjustment and a corresponding rms value. If the rms value is larger than 0 and not NA, then the constant adjustment is used as a fit instruments.

Method set_rms can be used to set individual rms values, while set_rms_values is a convenient method to give more than one rms value the same value.

Method get_rms returns all rms values larger than 0 and not equal to NA.

Usage

```
mdl$set_rms(values, name_err = c("warn", "stop", "silent"))

mdl$set_rms_values(value, names, pattern)

mdl$get_rms()
```

mdl is an IsisMdl object

Arguments

values A named numeric vector with rms values. The names should be the names of the corresponding frml variables (the variables at the left hand side of a frml equation).

name_err A character that specifies the action that should be taken when a name is not the name of a frml variable. For "warn" (the default), a warning is given, for "stop" an error is issued. For "silent", the variable is silently skipped,

value A numeric vector of length 1.

names A character vector specifying the names of the frml variables.

pattern A regular expression. The action (get or set rms values) is applied to the rms values corresponding to the frml variables with names matching pattern.

If neither names nor pattern has been specified in methods set_rms_values or get_rms, then the action is applied to all rms values.

Examples

```
mdl <- islm_mdl(period = "2017Q1/2018Q4")

mdl$set_rms(c(c = 5.0, t = 2, i = 21, md = 2))
print(mdl$get_rms())

# remove the constant adjustment for variable c from the fit instruments
```

```
mdl$set_rms_values(NA, "c")
print(mdl$get_rms())

# make all rms values equal to 1
mdl$set_rms_values(1)

# set the rms values for c and i to 2
mdl$set_rms_values(2, names = c("c", "i"))
```

`set_solve_options` `IsisMdl` *method: Sets the solve options*

Description

This method of R6 class `IsisMdl` can be used to set one or more solve options. These options will be stored in the `IsisMdl` object.

Method `get_solve_options` returns the solve options as a named list

Usage

```
mdl$set_solve_options(mode, fbstart, maxiter, maxjacupd, rlxspeed,
                      rlxmin, rlxmax, cstpbk, cnmtrx, xrelax,
                      xmaxiter, xupdate, dbgopt, erropt,
                      report, ratreport, ratreport_rep, ratfullreport_rep,
                      bktmax, xtfac, svdtest_tol)

mdl$get_solve_options()
```

`mdl` is an `IsisMdl` object

Arguments

All arguments below expect a numerical value unless mentioned otherwise.

`mode` a character string specifying the solution mode (`"auto"`, `"ratex"`, `"dynamic"`, `"reschk"`, `"backward"` or `"static"`). `"auto"` is the default. See section "Solution modes" below

`fbstart` a character string specifying the method of initializing feedback values. (`"current"`, `"previous"`, `"curifok"` or `"previfok"`). The default is `"current"`. See section "Feedback initialization methods" below

`maxiter` the maximum number of iterations per period (default 50)

`maxjacupd` the maximum number of Newton Jacobian updates per period (default 10)

`rlxspeed` Newton relaxation shrinkage (default is 0.5)

`rlxmin` Minimum Newton relaxation factor (default is 0.05)

`rlxmax` Maximum Newton relaxation factor (default is 1.0)

`cstpbk` Stepback criterion (default is 1.3). If the convergence criterion `Fcrit` is larger than `cstpbk` or invalid feedback variables have been obtained then the Newton step is not accepted and linesearching will be initiated. If the linesearching procedure failed (`Fcrit` is still larger than `cstpbk` after the maximum number of linesearch steps `bktmax` has been reached or if the relaxation factor has become smaller than `rlxmin`), a new Jacobean matrix is computed. In each linesearch step the current relaxation factor is shrunk by `rspeed`. The relaxation factor is set to its maximum value `rlxmax`) when a new Jacobian has been calculated.

`cnmtrx` Recalculate matrix criterion (default is 0.9). If the convergence criterion `Fcrit` is larger than `cnmtrx` but smaller than `cstpbk`, the Newton step is accepted but a new Jacobian is computed and the relaxation factor is set to its maximum value `rlxmax`. The new Jacobian is used in the next step. However, the Jacobian will not be recalculated if the number of Jacobian updates in a period is larger than `maxjacupd`

`xrelax` Rational expectations relaxation factor (default is 1)

`xmaxiter` Maximum number of rational expectation iterations (default is 10)

`xupdate` Character string defining the method of updating leads. Possible values are `"fixed"` (the default) and `"lastval"`. For `"fixed"` the leads beyond the solution period are fixed at the initial values. For `"lastval"` leads beyond the solution period take on the values from the last solution date

`dbgopt` A character vector specifying one more debugging options. See section "Debugging options" below

`erropt` Character string defining the error handling when invalid lags, leads, constant adjustments and/or exogenous variables are detected. Possible values are `"stop"` (stop on errors), `"cont"` (continue on errors but write a message to the output) and `"silent"` (also continue but without message). The default is `"stop"`

`report` A character string defining the type of computation progress report. Possible values are `"period"` (for a report per period), `"fullrep"` (for a full report), `"minimal"` (for a minimal report), and `"none"` (for no report). The default is `"period"`. The report options `"none"` also suppresses all output of the fit procedure and the Fair-Taylor progress report.

`ratreport` Defines the type of rational expectations progress report. See section "Ratex report options" below

`ratreport_rep` An integer number specifying the Fair-Taylor report repetition count. See Section "Ratex report options" below.

`ratfullreport_rep` An integer number, specifying the Fair-Taylor full report repetition count. See Section "Ratex report options" below.

`bktmax` Maximum number of backtracking linesearch steps with old Jacobian. Sometimes it is necessary for the Broyden method to take a shorter step than the standard step. This is called backtracking linesearch. `bktmax` is the maximum number of line search steps before a new Jacobian is computed.

`xtfac` Rational expectations convergence test multiplier When using the `"ratex"` solution mode, convergence of endogenous leads cannot be tested to the accuracy used in testing for convergence in the solution of the model. This option specifies the multiplier to apply to the convergence criterion for each endogenous variable if the variable has an endogenous lead. Suppose for example that some variable has a convergence criterion of 10^-5 and assume a value of 10 for the multiplier. Then its endogenous lead will be regarded as converged.

svdtest_tol Singular Value Decomposition (SVD) test tolerance parameter. If the inverse condition of the Jacobian is smaller than this parameter, then an SVD analysis of the Jacobian is performed. This may help to find the equations that cause (near) singularity of the Jacobian. The default value is -1, which implies that the SVD test is never performed. Specify a number between 0 and 1 to enable an SVD analysis depending on the inverse condition of the Jacobian. When this option has been specified a copy of the Jacobian is kept in memory, even if the Jacobian is not ill-conditioned. This option should therefore only be used during testing. It should be turned off in production calculations.

Solution modes

The solution mode can be specified with argument mode. Possible values are:

"auto" determine the solution mode automatically: "ratex" for models with endogenous leads and "dynamic" for models without endogenous leads

"dynamic" to update lags and current values of all right-hand side endogenous variables (leads are not updated). This is the default for models without endogenous leads

"ratex" to update lags, leads and current values of all right-hand side endogenous variables. This is the default mode for models with endogenous leads. The model is solved in dynamic mode for all periods conditional on the endogenous right-hand side leads. After solving for the complete solution period the endogenous leads are updated with the the results for the corresponding endogenous variables. The solution process thus consists of an two loops: an inner loop solving the model for all periods given the leads and an outer loop which solves for the endogenous leads

"static" to update only current values of right-hand side endogenous variables lags and leads are not modified

"reschk" to not update right-hand side endogenous variables from the solution

"backward" same as dynamic, except that the model is solved backwards. The model is solved in reversed order by starting at the last solution period and ending at the first solution period. Leads are updated and lags are not updated

If a model contains leads then the "ratex" mode is the default; this is a Fair-Taylor algorithm.

The default is "auto".

Feedback initialization methods

Argument fbstart can be used to specify the way how the feedback variables at the current period (i.e. the period for which the model is being solved) are initialized from the model data. Possible values of fbstart are:

"current" the initial values are always taken from the current period. This is the default

"previous" The initial values are taken from the previous period except when the first period to be solved is the start of the model data period. In that case current period values are used

"curifok" Current period values are used if they are valid otherwise previous period values are used

"previfok" At the start of the solution period, previous period values will be used if they are available and valid; otherwise current period values will be used. Thereafter previous period initial values are always used, which is equivalent to the "previous" method

The default is "current".

Debugging options

Argument dbgopt can be used to specify one or more options for debugging. Possible values are

"prifb" print feedback variables at each iteration

"prild" print all leads at each ratex iteration

"prijac" print Jacobian matrix when updated

"prinoconv" print all not converged endogenous variables

"prinotconvl" print all not converged leads

"allinfo" all of the above

"noprifb" do not print feedback variables at each iteration

"noprild" do not print all leads at each ratex iteration

"noprijac" do not print Jacobian matrix when updated

"noprinoconv" print only the largest discrepancy of all not converged endogenous variables

"noprinotconvl" print only the largest discrepancy of all not converged leads

"noinfo" no debugging output

"priscal" print scaling factors as determined from the Jacobian

nopriscal do not print scaling factors as determined from the Jacobian

Default is no printing of debugging information.

Ratex report options

The type of report is determined by argument ratreport. Arguments ratreport_rep (the report repetition count) and ratfullreport_rep (the full report repetition count), both specified as integer numbers, can be used to further modify the progress report.

Possible values for ratreport are

"iter" print the number of not converged expectation values every ratreport_rep Fair-Taylor iteration (the default)

"fullrep" full report. The number of not converged expectation values is printed every ratreport_rep Fair-Taylor iteration and the largest remaining discrepancy every ratfullreport_rep Fair-Taylor iteration

"minimal" for a full report only after the last Fair-Taylor iteration

If ratfullreport_rep is NA, then the full report is printed every ratreport_rep Fair-Taylor iteration. The default values for ratreport_rep and ratfullreport_rep are 1 and NA, respectively.

See Also

set_debug_eqn

Examples

```
mdl <- islm_mdl(period = "2017Q1/2018Q4")
mdl$set_solve_options(maxiter = 100)
```

set_user_data IsisMdl *method: Set and get user data*

Description

An IsisMdl object is equipped with a list with user data. This list is empty by default. With method set_user_data of R6 class IsisMdl elements can be added to this list. Method get_user_data() returns the user data.

Usage

```
mdl$set_user_data(user_data, ...)

mdl$get_user_data(key)
```

mdl is an IsisMdl object

Arguments

user_data The user data. This should be a named list.

... The other arguments passed to set_user_data are used to update the user data list. See the examples.

key A character specifying the key(s) of the user data elements to retrieve. If not specified the complete user data list is returned.

Function get_user_data returns a single element of the user data list if argument key has been specified and if this is a single character; otherwise the function returns a list.

To remove an element of the list, set it to NULL (see example)

Examples

```
mdl <- islm_mdl(period = "2021q1/2021q2")

mdl$set_user_data(date = Sys.Date(),
                  note = "Example of user data")

# the previous statement is equivalent to:
mdl$set_user_data(list(date = Sys.Date(),
                       note = "Example of user data"))

# add another user data element
mdl$set_user_data(input_data = mdl$get_data())

# print all user data
print(mdl$get_user_data())

# print a specific element of the user data
print(mdl$get_user_data("date"))
```

```
# print two specific elements of the user data
print(mdl$get_user_data(c("date", "input_data")))

# remove user data element 'input_data':
mdl$set_user_data(input_data = NULL)
print(mdl$get_user_data())
```

`set_values-methods` *IsisMdl methods: Sets the values of the model data, constant adjustments, fix values or fit targets*

Description

These methods of R6 class `IsisMdl` can be used to set the values of the model data, constant adjustments, fix values or fit targets.

Usage

```
mdl$set_values(value, names, pattern, period = mdl$get_data_period())

mdl$set_ca_values(value, names, pattern, period = mdl$get_data_period())

mdl$set_fix_values(value, names, pattern, period = mdl$get_data_period())

mdl$set_fit_values(value, names, pattern, period = mdl$get_data_period())
```

`mdl` is an `IsisMdl` object

Arguments

`value` A numeric vector of length 1 or with the same length as the length of the range of `period`.

`names` A character vector with variable names. For `set_ca_values` and `set_fix_values`, the names should be the names of frml variables (the variables on the left hand side of frml equations). For `set_fit_values`, the names should be the names of endogenous variables.

`pattern` A regular expression specifying the variable names.

`period` A `period_range` object or an object that can be coerced to a `period_range`. The default is the data period. The frequency of `period` should be equal to or lower than the frequency of the model period. If the frequency is lower, than the period range is converted to the same frequency as the model frequency with function `change_frequency` of package `regts`.

If neither `names` nor `pattern` has been specified, then the action is applied to all model variables of the appropriate type.

Methods

`set_values` Sets the values of model data.

`set_ca_values` Sets the values of the constant adjustments, i.e. the residuals of 0 (frml) equations.

`set_fix_values` Set fix values for the stochastic model variables (i.e. model variables that occur at the left hand side of a frml equation). The model variables will be fixed at the specified value. A fix value of `NA` implies that the corresponding variable is not fixed. `set_fix` also updates the model data with all non NA values.

`set_fit_values` Set fit targets for the fit procedure. A fit target value of `NA` implies that the corresponding variable is no fit target.

See Also

`get_data-methods`, `set_data-methods` and `change_data-methods`

Examples

```
mdl <- islm_mdl(period = "2017Q1/2017Q3")

# set the values for y in the full data period
mdl$set_values(1000, names = "y")

# set the values of ms and md in 2017Q1 and 2017Q2
mdl$set_values(c(205, 206), pattern = "^m.$", period = "2017Q1/2017Q2")

# set the values of ms and md in all quarters of 2017 (2017Q1/2017Q4)
mdl$set_values(c(205, 206, 207, 208), pattern = "^m.$", period = "2017")
print(mdl$get_data())

# give the constant adjustment of variable c the value 1
mdl$set_ca_values(0, names = "c")

# fix c and i at 200 in period 2017q1/2017q2
mdl$set_fix_values(200, names = c("c", "i"))
```

`solve` `IsisMdl` *method: Solves the model.*

Description

This method of R6 class `IsisMdl` solves the model. It requires that the model period has been set with methods `isis_mdl`, `init_data` or `set_period`).

Usage

IsisMdl method:

```
mdl$solve(period = mdl$get_period(), options = list(),
          fit_options = list()
```

mdl is an IsisMdl object

Arguments

period period_range object, or an object that can be coerced to period_range

options a named list with solve options, for example list(maxiter = 50). The names are the corresponding argument names of method set_solve_options The specified options will only used in this call of solve() and will not be stored in the IsisMdl object

fit_options a named list with options for the fit procedure, for example list(maxiter = 10). The names are the corresponding argument names of method set_fit_options The specified options will only used in this call of solve() and will not be stored in the IsisMdl object

Details

The model will be solved for each subperiod from the solution period sequentially. The solution is stored in the IsisMdl object, and can be retrieved by methods get_data (or get_ca for the constant adjustments). Any subsequent solves of a model will use these data. If a solve has converged and no data have changed, then a second solve will report convergence in 0 iterations.

The solve options specified are only applied to the current solve. If none are specified the solve options as specified with method set_solve_options are used.

The solve procedure *never* raises an error, even if the solve was not successful. In that case a warning may be issued. It is up to the user to perform any checks. Method get_solve_status can be used to check whether the solve was successfully terminated or not. The solve method outputs a report which the user should check.

See Also

set_solve_options, set_fit_options and get_solve_status

Examples

```
mdl <- islm_mdl(period = "2017Q1/2018Q4")
mdl$solve(options = list(report = "fullrep"))

# solve the model for all periods before 2018Q1
mdl$solve(period = "/2017Q4")

# solve the model for all quarters in 2017 (2017Q1/2017Q4)
mdl$solve(period = "2017")
```

`solve_mdl` *Function solve_mdl solves model for given data and returns resulting data and constant adjustments*

Description

Function solve_mdl solves model for given data and returns resulting data and constant adjustments

Usage

```
solve_mdl(model_file, data, period, fix_values, ca, fit_targets)
```

Arguments

`model_file`	is a reference to the file containing the `IsisMdl` model
`data`	is a `regts` object containing time series data
`period`	is a `period` object describing a time interval
`fix_values`	is a `regts` object containing known time series data that should be fixed during analysis
`ca`	describes the so-called constant adjustment values
`fit_targets`	describes the so-called fit targets

See Also

`set_data-methods`,`get_period`

`write_mdl` *Writes an* `IsisMdl` *object to a file*

Description

This method of R6 class `IsisMdl` serializes the model object and writes it to a binary file. The model can be read back by function `read_mdl`.

Details

`write_mdl` employs the serialization interface provided by base R function `saveRDS`.

Usage

```
mdl$write_mdl(file)
```

`mdl` is an `IsisMdl` object

Arguments

`file` the filename. Preferably use the extension `.ismdl` so that it is obvious that the written file contains a serialized `IsisMdl` object.

See Also

`read_mdl`

Examples

```
mdl <- islm_mdl("2017Q1/2019Q2")
mdl$write_mdl("islm_mdl.ismdl")
```

Index